(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72574 A2

(51) International Patent Classification[7]: H04N

(21) International Application Number: PCT/US00/13882

(22) International Filing Date: 17 May 2000 (17.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/316,328 21 May 1999 (21.05.1999) US

(71) Applicant: QUOKKA SPORTS, INC. [US/US]; 525 Brannan Street, San Francisco, CA 94107 (US).

(72) Inventors: RAMADAN, Alan, S.; 151 Lark Lane, Mill Valley, CA 94941 (US). SUSSNA, Jeffrey, E.; 903 Ventura Way, Mill Valley, CA 94941 (US). BROCCHINI, Matthew, Alan; 2305 Adeline Drive, Burlingame, CA 94010 (US). SCHAEFER, William, B., IV; 559 46th Avenue, San Francisco, CA 94121 (US). TAYLOR, John; 28A Lower Crescent Avenue, Sausalito, CA 94965 (US).

(74) Agents: MALLIE, Michael, J. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th Floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 00/72574 A2

(54) Title: AN ARCHITECTURE FOR CONTROLLING THE FLOW AND TRANSFORMATION OF MULTIMEDIA DATA

(57) Abstract: A method and apparatus for controlling the flow of information from the event is described. In one embodiment, the method comprises receiving digital media assets corresponding to remotely captured data from the event, converting the digital media assets into immersive content, and distributing the immersive content for delivery to a plurality of delivery mechanisms.

AN ARCHITECTURE FOR CONTROLLING THE FLOW AND TRANSFORMATION OF MULTIMEDIA DATA

FIELD OF THE INVENTION

The present invention relates to the field of broadcasting events using multiple media; more particularly, the present invention relates to controlling the acquisition, production, flow, and presentation of multimedia data using a structure such as a context map to indicate relationships between event data and contexts.

BACKGROUND OF THE INVENTION

Television has been the traditional medium for broadcasting certain events, such as sporting events, to consumers. Consumers have long been limited to those broadcasts which the broadcasters believe are worthy of viewing. Moreover, even with respect to individual events, broadcasters decide which portions of a particular event to broadcast.

Individual viewers may wish to view events through a perspective that is different from that of the broadcasters. For instance, if a broadcaster is showing a sporting event on television, an individual viewer may wish or desire to follow an individual competitor, sponsor, etc. However, the individual viewer does not have control over the particular content that is broadcasted by the broadcaster and cannot indicate the content they desire to see as an event is being broadcast.

In addition, current broadcasting technologies are unable to organize and transmit the rich diversity of experiences and information sources that are available to participants or direct spectators at the event. For example, a live spectator or participant at a sporting event is able to simultaneously perceive a wide range of information, such as watching the event, listening to it, reading the program, noticing weather changes, hearing the roar of the crowd, reading the scoreboard, discussing the event with other spectators, and more. Thus, the spectator/participant is immersed in information relating to the event. Furthermore, a knowledgeable spectator knows how to direct his attention within this flood of information to maximize his experience of the event. The viewer of a television broadcast does not and cannot experience this

information immersion. That is, television lacks the ability to emulate for the viewer the experience of attending or participating in the particular event.

In order to provide a viewer virtual emulation of the event (i.e., to enable an individual to be immersed in information relating to the event) while the viewer is not a live spectator, what is needed is a way to deliver this rich set of experiences and information (which may include multiple video, audio, text, image, data telemetry, biometrics, weather, chat, and other kinds of data and interactivity), so that individual viewers can simultaneously perceive and interact with many of them, freely directing their attention among them, and so that a production team can provide guidance that will help less knowledgeable spectators direct their attention to the information sources that provide the best experience of the event at any given moment.

SUMMARY OF THE INVENTION

A method and apparatus for controlling the flow of information from an event is described. In one embodiment, the method comprises receiving digital media assets corresponding to remotely captured data from the event, converting the digital media assets into content by using a context map to organize the digital media assets and indicate relationships between contexts associated with the digital media assets, and distributing the content for delivery to a plurality of delivery mechanisms.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the invention, which, however, should not be taken to limit the invention to the specific embodiments, but are for explanation and understanding only.

Figure 1 illustrates one embodiment of a platform.

Figure 1A illustrates the data flow through one embodiment of the platform.

Figure 2 is an alternative view of the platform.

Figure 3 illustrates one embodiment of the subprocesses of production.

Figure 4 illustrates a simple route allocated to a flow-through video stream.

Figure 5 illustrates an example of a context defined using metadata.

Figure 6 is a context depicted to facilitate illustrating the use of URN's and URN references.

Figure 7 illustrates one embodiment of an architecture for an end user client.

Figure 8 illustrates exemplary asset groups, stylesheets and presentation modules.

Figure 9 illustrates an exemplary stylesheet describing a layout and presentation.

Figure 10 illustrates a simple context map.

Figure 11 is a schematic of a control flow.

Figure 12 is a block diagram of one embodiment of an architecture for a client.

DETAILED DESCRIPTION

A platform, or architecture, that uses a group of processes and mechanisms to deliver digital media assets from events (e.g., sporting events) and/or other sources to end users is described. The digital media assets include assets (individual units of digital content) and metadata.

An asset may include, for example, material such as a photographic image, a video stream, timing data from an event (e.g., timing data from a race, timing data for a single car on a single lap within a race), the trajectory of a ball as it flies towards a goal, an HTML file, an email (from a computer), etc.

Metadata is information about an asset, such as for example, its type (e.g., JPEG, MPEG, etc.), its author, its physical attributes (e.g., IP multicast addresses, storage locations, compression schemes, file formats, bitrates, etc.), its relationship to other assets (e.g., that a photograph was captured from a given frame of a particular video),

the situation in which an end user accessed it (e.g., a hit), its heritage (e.g., other assets from which it was generated), its importance to the immersive experience, and its movement through the platform. In one embodiment, metadata may provide more abstract information, such as for example, the types of values available within a particular kind of telemetry stream, instructions generated by production and followed by immersion (e.g., to track certain kinds of user behavior, to automatically present certain assets to the user at certain times or in response to certain user actions, etc.), relationships between assets and other entities such as events, competitors, sponsors, etc. In one embodiment, the platform treats both assets and metadata as first-class objects, which are well-known in the art.

Data flow and context management are critical to operation of the platform. Data flow refers to the transfer of information, while context management refers to controlling the relationship between data. In one embodiment, a context is a metadata structure that defines a set of assets (and/or other contexts). Contexts may be dynamically generated or may be stored for optimal access.

The platform controls the data flow from each event. The platform collects assets from a venue, produces an immersive experience and delivers the experience to end users (viewers). In one embodiment, the platform receives digital media assets (e.g., metadata and individual units of digital content) corresponding to the event and converts those assets into immersive content (i.e., context from and about an event in which users may immerse themselves).

In one embodiment, the conversion of digital media assets into immersive content is performed by using a context map (e.g., a graph structure), which organizes the digital media assets and indicates relationships between contexts associated with the digital media assets. The platform may maintain a hierarchical database of contexts. As part of using a context map, the platform tags digital media assets with global identifiers indicative of context information. In one embodiment, the global identifier is a persistent, location-independent, globally unique identifier to the digital media asset describing the event.

The immersive content is distributed, delivered, and presented to end users (viewers). The immersive content enables an immersive experience to be obtained. This immersive experience is a virtual emulation of the experience of actually being

present at or participating in an event, obtained by being subjected to the content that is available from and about the event.

Thus, the platform collects, transmits, produces, distributes, delivers, and presents the digital media assets. Each of these functions, or phases, may be implemented in hardware, software, or a combination of both. In alternative embodiments, some of these may be performed through human intervention with a user interface.

In the following description, numerous details are set forth. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system

memories or registers or other such information storage, transmission or display devices.

The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

The instructions of the programming language(s) may be executed by one or more processing devices (e.g., processors, controllers, control processing units (CPUs), execution cores, etc.).

An Exemplary Platform

Figures 1 and 2 illustrate the platform in terms of six subprocesses or segments which it performs: collection 101, transmission 102, production 103, distribution 104, delivery 105 and immersion 106. In the following description, each of these will be discussed briefly and then in more detail.

Referring to Figure 1, collection 101 is a segment of the platform that collects data of an event. In one embodiment, collection 101 of assets occurs at the venue of the event. In an alternative embodiment, raw assets from the event are delivered directly to a studio where collection 101 occurs.

The event may be any type of event, such as, for example, a sporting event, a concert, etc. Examples of sporting events include, but are not limited to, an auto race, a mountain climbing expedition, a fleet of boats spread across an entire ocean, etc.

In one embodiment, collection 101 gathers the digital assets from which an immersive content will be created (video, text, images, audio, etc.) and packages (e.g., converts) them into a format. In one embodiment, a specific format is used to communicate assets, context, and other metadata and will be described in more detail below. A single format may be used to communicate all assets, context and other metadata, including time information indicative of when the asset was created or collected. The output of collection 101 comprises digital media assets.

Transmission 102 transmits digital media assets to a studio using communications facilities (e.g., satellites, etc.). In one embodiment, assets are transmitted in a specific format over Internet Protocol (IP) networks.

Production 103 converts digital media assets into immersive content. In one embodiment, production 103 includes subprocesses that may be performed on digital media assets at a studio, such as, for example, video editing, HTML writing/page creation, automatic processing of telemetry, creation of views (or groups of digital media assets), stylesheets, display modules, setting view priorities, managing a view queue or other structure containing display information, etc. In one embodiment, almost all of the production work occurs at the studio. Production 103 may produce different distribution streams for different types of delivery. For instance, a satellite system may include high resolution video streams, while basic internet delivery will not. Therefore, depending on the type of delivery for the content being produced, one or more different distribution streams may be created at production.

Distribution 104 transmits content from the studio to one or more delivery media devices, channels and/or systems, which forwards the content to individual end users (e.g., an audience). Distribution 104 may use a distribution network to distribute different streams generated by production 103 to different types of delivery mechanisms that deliver the content to end users (e.g., the audience). Such a distribution may depend on the communications mechanism available to end users.

Delivery 105 delivers the content to the audience using one or more types of media (e.g., satellite, cable, telco, network service provider, television, radio, on-line, print, etc.).

Immersion 106 is a segment of the platform where content is received and presented to end users for viewing and interacting. In one embodiment, the end users generate audience intelligence (AI) information that is sent back to production 103 for analysis.

Figure 1A illustrates the data flow through one embodiment of the platform. Referring to Figure 1A, collection processing 191 generates streams and packages that are forwarded, via transmission processing 192, to production processing 193. A stream is a constant flow of real time data. A package is a bundle of files that is stored and forwarded as a single unit. A stream resembles a radio or television program, while a package resembles a letter or box sent through the mail (where the letter or box may contain video or audio recordings). Streams are often used to allow end users to view time-based content (e.g., real-time video data), while packages may be used for non-temporal content (e.g., graphics, still images taken at the event, snapshots of content that changes less rapidly than time-based content, such as a leaderboard, etc.). In one embodiment, the streams and packages being transmitted are formatted by collection processing 191 into a particular format.

The streams and packages flow through the platform from collection processing 191 through to immersion processing 196. Collection processing 191 and production processing 193 exchange metadata in order to synchronize contextual information. From the production processing 193 through the delivery processing 195, streams, packages and metadata are transferred via distribution processing 194. Between delivery processing 195 and immersion processing 196, the streams and packages may be converted into formats specific to the delivery technology (e.g., http responses, etc.). AI information is fed back from immersion processing 196 and delivery processing 195 to production processing 193. The AI information may include user identification and activity information. In one embodiment, the AI information is represented using the same format as the streams and packages and metadata. The AI information fed back from delivery processing 195 may be information that is sent on behalf of immersion processing 196 and/or may be sent because immersion processing 196 is not capable of sending the information. For example, a hit log for a web page may be sent back from a server in delivery processing 195. Monitoring and control processing 197 controls the processes throughout the platform and monitors the operation of individual segments.

Each of the six segments, or processes, are discussed in detail below.

Detailed Discussion of Platform Sub-processes

Collection 101 is the process of capturing proprietary data at event venues and translating it into a format. That is, at its most upstream point, the platform interfaces to a variety of data acquisition systems in order to gather raw data from those systems and translate it into a predefined format. The format contains media assets, context and other metadata. In one embodiment, the format adds a global identifier, as described in more detail below, and synchronization information that allows subsequent processing to coordinate content from different streams or packages to each other. The processes within collection 101 are able to control gathering and translation activities.

In one embodiment, collection 101 converts raw venue data into digital media assets. The venue data may include both real-time and file-based media. This media may include traditional real-time media such as, for example, audio and video, real-time data collected directly from competitors (e.g., vehicle telemetry, biometrics, etc.), venue-side real-time data (e.g., timing, position, results, etc.), traditional file-based media (e.g., photographs, editorial text, commentary text, etc.), other file-based media (e.g., electronic mail messages sent by competitors, weather information, maps, etc.), and/or other software elements used by the client (e.g., visualization modules, user interface elements dynamically sent out to client to view data in new ways, sponsor (advertising) contexts, view style sheets).

Sporting events can be characterized by the assets collected from the event. In one embodiment, these assets can include audio and video of the actual event, audio or video of individual competitors (e.g., a video feed from an in-car camera), timing and scoring information, editorial/commentary/analysis information taken before, during and/or after an event, photographs or images taken of and by the competitors, messages to and from the competitors (e.g., the radio links between the pit and the driver in an auto race), a data channel (e.g., a control panel readout taken from the device in a car), telemetry indicating vital functions of a competitor. In one embodiment, the telemetry can include biometrics of the competitor (e.g., heart rate, body temperature, etc.). Other telemetry may include position information of a competitor (e.g., player with microchip indicating position) using, for example, a global positioning system (GPS), telemetry from an on-board computer, etc., or a physical device (e.g., an automobile).

Various devices may be used to perform the collection of sporting event data, or other data. For example, cameras may be used to collect video and audio. Microphones may be used to collect audio (e.g., audience reaction, participant reaction, sounds from the event, etc.). Sensors may be used to obtain telemetry and electronic information from humans and/or physical objects. The information captured by such devices may be transferred using wires or other conductors, fibers, cables, or using wireless communications, such as, for example, radio frequency (RF) or satellite transmission.

In one embodiment, collection 101 includes remote production. Remote production is the process of managing an event from a particular point of view. In one embodiment, managing an event includes determining which assets will be collected and transferred from the event venue. In one embodiment, event management includes: defining, statically or dynamically, event-specific metadata based on global metadata received from production; dynamically controlling which assets are captured (using dynamic selection of information as event data is being collected), how they are formatted (e.g., adjusting a compression rate using a video encoder depending on contents of the video), and transmitted away from the event; managing physical resources (data collection hardware, communications paths and bandwidth, addresses, etc.) necessary to capture, format, and transmit assets; locally producing assets (e.g., editorial text) to complement those being captured; and generating metadata in order to transmit event-specific metadata definitions back to production 103 (while performing production typically, but possibly while running the event).

A remote production facility (RPF) performs the remote production and allows event producers to manage venue-side production (i.e., located at the event) processes for an event. In one embodiment, the RPF contains one or more graphical user interfaces (GUIs) for controlling one or more of the following functions: bandwidth management (allocating individual chunks of transmission bandwidth), stream control (dynamically allocating physical resources in order to capture, process, and transmit specific data streams), limited metadata management (a subset of the functionality supported by the studio), limited asset production (a subset of the functionality supported by the studio), and limited immersion production (a subset of the functionality supported by the studio).

Certain events, however, do not support co-located remote production (for example, a climbing expedition on an isolated mountain). In such cases, the RPF is positioned further from the venue, or is omitted from the process and all RPF functions are performed at the studio.

In one embodiment, collection 101 uses includes hardware, such as collection devices or data acquisition systems (e.g., cameras, microphones, recorders, sensors, etc.), communications equipment, encoding servers, remote production management server(s), and network equipment. In one embodiment, each of the collection devices converts the event data it captures into a format that includes the digital units of data, metadata and context information. In an alternative embodiment, each of the captured devices sends the raw captured data to a location where a remote production unit, device, or system formats it.

In one embodiment, a data acquisition application programming interface (API) provides a uniform interface between data acquisition systems and a remote production system. This allows developers to write specific gathering and translation modules as plug-ins for interfacing the data acquisition systems to the remote production system that ensure that the remote production system can handle the data.

Transmission

Transmission 102 transmits specifically formatted streams and packages, including metadata, from event venues to a studio. In one embodiment, streams and packages are transferred via high speed IP networks. Depending on the event location, the IP networks may be terrestrial and/or satellite-based. A communication mechanism of transmission 102 for the transmission of streams may be selected based on its ability to accommodate bandwidth management, while a communication mechanism of transmission 102 for the transmission of packages may be selected based on its reliability. In either case, transmission 102 treats the specifically formatted assets as opaque entities. In other words, transmission 102 has no knowledge of what data is being transmitted, nor its format, so that the information is just raw data to transmission 102.

Depending on the underlying transport technology, transmission 102 may include dynamic network provisioning for individual sessions. That is, the network may dynamically allot more bandwidth to particular streams or packages based on

priority. Data could be routed over links based on cost or time priorities. For example, transmission 102 may purchase transport bandwidth, while a terrestrial IP network is on all the time. Supplementary data might be routed over Internet virtual private networks while video might be sent over a satellite.

In one embodiment, transmission 102 may include a process that encrypts assets prior to transmission.

Production

Production 103 includes a set of processes that are applied to digital media assets before they are distributed. The digital media assets include specifically formatted streams and packages resulting from collection 101, as well as content being produced within the process of production 103 itself. Production 103 uses a studio as a central site where the digital media assets are produced before being distributed. In one embodiment, the studio is an internet protocol (IP) studio. It is referred to as an IP studio because all, or some portion of, digital media assets that are received from the studio are sent out using an industry standard TCP/IP protocol suite throughout the rest of the segments (phases) and the assets are digital IP assets. The studio may not send and view the digital video assets or perform all operations using IP in alternative embodiments.

In one embodiment, the studio fundamentally operates as a switch in which formatted content from multiple sources is received and sent out to multiple destinations. Numerous operations may be performed on the content, such as archiving, compression, editing, etc., as part of the switching process. That is, there is no hardwired connection between the operations and they may be performed on the pool of assets in general. Other production operations may be performed by the studio such as, for example, laying out text and graphics, setting priorities for views (assets groups), creating associations between assets, etc.

In one embodiment, production 103 comprises the following processes: acquisition, asset storage, asset production, analysis, immersion production, metadata management, dissemination, process management, user management, distillation and syndication. In one embodiment, each of these operate in a manner decoupled from each other. The process may be implemented as hardware and/or software modules. Figure 3 illustrates each of these processes.

Referring to Figure 3, the acquisition process 301 provides the interface between transmission 102 and production 103. Acquisition process 301 receives specifically formatted streams, packages, and metadata from collection 101 and parses them into assets (units of digital data) and metadata. Metadata may come to the acquisition process 301 separate from digital media assets when the metadata cannot be attached to event data that has been captured. This may be the case with an NTSC-based stream of data, where in such a case the metadata may indicate that the stream is an NTSC stream.

In one embodiment, the acquisition process 301 provides an interface through which a number of operations may be performed. For instance, in one embodiment, the acquisition process 301 decrypts assets that had been encrypted for secure transmission, unpackages packages into their constituent parts, parses metadata messages to determine their type and meaning, collects AI usage and user identification information flowing back to production 103 from delivery 105 and immersion 106 and/or logs arrival times for all assets.

The acquisition process 301 parses the metadata messages from the information received and forwards their contents to the metadata management process 306. After initially processing assets, the acquisition process 301 forwards them to the asset storage process 302. It also registers new assets with the metadata management process 306. The registration may be based on a context map that indicates what assets will be collected, and the tag on each asset (attached at collection). During the acquisition process 301, using the tag, the process knows one or more of the following: what to do with the asset (e.g., store for later use, pass through unchanged, contact context management to notify it that the asset has been received, etc.).

Some assets, particularly streams, flow in and out of the studio with as little latency as possible. In this case, the acquisition process 301 forwards the assets directly to the dissemination process 307. In one embodiment, flow-through assets are simultaneously forwarded from the acquisition process 301 to the asset storage process 302 and the dissemination process 307. This is shown in Figure 4.

The asset storage process 302 manages physical storage and retrieval of all assets. One or more storage media may be used. In one embodiment, a variety of storage technologies may be used, each suitable for certain types of assets.

In one embodiment, the asset storage process 302 is responsible for interacting with the appropriate storage technology based on asset types. A given asset type may be stored in multiple ways. For example, a telemetry stream may be stored as a flat file in memory and as a set of database records. In one embodiment, the asset storage process 302 is involved in storage, retrieval, removal, migration and versioning. Migration refers to moving assets up and down within a storage hierarchy. This movement may be between different storage technologies (e.g., between hard disk and tape). Migration may be performed to free up local or short-term storage. Versioning may be used to indicate an asset's current version (after changes to this asset have been made or have occurred). In one embodiment, every time the asset storage process 302 stores, removes, migrates, or versions an asset, it communicates with the metadata management process 306 to update the asset's physical location attributes, which the metadata management process 306 manages.

The asset production process 303 is the set of processes by which individual digital media assets are created and edited within production 103. The asset production process 303 is applied to most assets that have been acquired from collection 101. In addition, in one embodiment, in-house editorial and production staffs may create and edit their own assets during the asset production process 303. In one embodiment, asset production process 303 includes creation, editing, format conversion (e.g., Postscript to JPEG, etc.), and distillation.

A number of editing tools may be used. In one embodiment, the creation and editing processes are performed in cooperation with the asset storage process 302 and the metadata management process 306. This interaction may be automatic or manual. In one embodiment, assets are transferred from asset storage in order to be edited, and transferred back into asset storage after editing has been completed. In addition, the effects of production actions are communicated to the metadata management process 306. For example, the asset production process 303 notifies the metadata management process 306 that a JPEG asset was derived from a Postscript asset.

The distillation process creates multiple versions of an asset to support different kinds of delivery technologies (e.g., high, medium, and low-bandwidth web sites, one-way satellite data broadcast, interactive television, etc.). The distillation process is performed by assessing the capabilities of the delivery technology against the asset and type of data being transformed. Depending on the complexity of the differences, the

distillation process may be more or less automated. In any case, in one embodiment, the distillation process takes into account many aspects of delivery, including, but not limited to, file format, and the number and kind of assets that will be included for a specific delivery platform.

The analysis process 304 coordinates the AI user activity and identification information flows into the production process via the acquisition process 301 and asset storage processing 302. The AI information may indicate user requests as part of their activity information. In one embodiment, the analysis process 304 directs AI data to a data warehouse (or other storage). Once in the warehouse, this information can be analyzed using tools (e.g., data mining tools). This analysis may be done either manually or automatically.

Immersion production process 305 attaches greater meaning to the assets that flow through production. In one embodiment, the immersion production process 305 initially creates HTML pages that reference and embed other assets. In one embodiment, the immersion production process 305 also creates and edits contexts, generates AI instructions to indicate to immersion applications which user behavior to track and/or what actions to take when this behavior occurs, generates (manually or automatically) one kind of content based on another (e.g., highlight generation, running averages derived from telemetry values, specifically formatted metadata based on context management, etc.), generates production instructions directing immersion applications to automatically present certain information based on specific user actions, uses immersion applications to view content for quality control purposes, and defines the types of values available within particular stream types.

The metadata management process 306 manages metadata within production 103. Metadata may include many different types of data. In one embodiment, metadata includes asset attributes, production instructions and contexts. The production instructions may control the immersion applications based on activity of the user. In one embodiment, all types of metadata are first-class objects, thereby allowing easy transport between the platform segments. In one embodiment, every object has a unique identifier and a set of attributes. Unique identifiers are used to track objects and to relate them to other objects.

The metadata management process 306 creates and modifies attributes and contexts, logically locates objects by querying contexts (e.g., locate all streams belonging

to Zinardi's car in the Long Beach race), logically locates objects by querying asset attributes (e.g., locate all the JPEG assets whose author is "Emily Robertson"), physically locates objects by tracking their movements in the form of asset attributes (e.g., version #5 of asset #456 has been stored in the file named "foo.JPG.5" on the file server named "file_server_1").

The dissemination process 307 provides the interface between production 103 and distribution 104. To facilitate this interface, the dissemination process 307 is configured to communicate with individual distribution channels. The dissemination process 307 communicates with the asset storage process 302 to retrieve assets and with the metadata management process 306 to retrieve metadata. The dissemination process 307 also communicates directly with the acquisition process 301 in the case of flow-through streams. In one embodiment, the dissemination process 307 provides an interface for a number of operations. In one embodiment, the dissemination process 307 provides an interface that constructs messages out of metadata, packages assets and metadata into packages, optionally encrypts data for secure distribution, and logs departure times for all streams and packages.

When the studio generates content for distribution, the dissemination process 307 sends the digital media assets to various delivery head ends. In one embodiment, the type of data that is distributed to different types of devices is dependent on the device and the dissemination process 307 controls which streams and packages of data are forwarded to the delivery head ends. In one embodiment, a device such as a Personal Digital Assistant (PDA) will only be sent data that it is capable of displaying. Similarly, an HDTV device will only be sent data that it will capable of displaying. In an alternative embodiment, all data that is available is forwarded to the device and the device makes a determination as to whether it can or cannot display some or all of the information. In one embodiment, to notify a particular device of the type of data that is being sent to it, the studio distributes a control stream. In one embodiment, this control stream is the context map. That is, the context map is sent to the end user devices. In an alternative embodiment, only a portion of the context map that specifically deals with the event being captured is forwarded to the device to indicate what types of digital media assets are being forwarded. Based on the information in the control stream, the end user devices may determine what information is being sent to it and may determine what to view.

The process management 308 is a process that controls the automation of other production processes. The process management 308 uses several types of objects to control asset switching (routing). In one embodiment, these types of objects include routes, process events, schedules and rules. A route is a mapping between a set of processes and a set of physical (e.g., hardware, software, and network) resources. For example, Figure 4 illustrates a simple route allocated to a flow-through video stream. Referring to Figure 4, a stream is received from an incoming network 401 and undergoes acquisition via the acquisition process 301. From acquisition, the video stream is forwarded to both the assets storage process 302, which stores the video stream and makes it accessible on a video server 403, and the dissemination process 307 where the video stream is disseminated to an outgoing network 402.

A process event is the application of a given route to a particular asset or group of assets at a specific time. A schedule is the set of times at which a processing event occurs. A rule is a logical constraint that determines when an event occurs. For example, a rule might state that a static leaderboard update page should be generated whenever a leaderboard stream has been acquired and archived. By using these objects, the assets may be managed, including indicating what information is to be shown.

The process management 308 also provides an interface for creating, querying, and editing routes, process events, schedules, and rules. In one embodiment, the process management 308 also keeps a log of every completed event and the success or failure of its outcome.

The user management process 309 controls access by production users to the various processes within the studio. In one embodiment, the user management process 309 manages definitions of users, groups, and access levels. Based on these definitions, it responds to requests from the process management 308 to provide access credentials for particular studio activities.

The syndication process 310 allows 3rd-party organizations (e.g., external media companies) access to assets within the studio. In one embodiment, individual assets and subscriptions can be offered, with e-commerce taking place based on those offers.

As mentioned above, processes of production 103 occur in a studio. The studio contains a hierarchical or other type of arrangement of asset storage hardware and software (e.g., a database, robotic-type system, etc.). The asset storage control system controls the flow of assets up and down within that hierarchy and determines how to

route assets based on their types. In a live environment, the asset storage system would direct data of differing types (e.g., telemetry vs. video) to appropriate storage types. The asset storage system can also make intelligent decisions about asset migration, for example, based on the time since the asset was accessed, the relationship of the asset to current production activity (as determined by context analysis, the time-sensitivity of the asset, and/or an industry standard algorithm (e.g., least recently used (LRU)). As the production evolves, the asset system might choose to push some (previously used) data into off-line storage (e.g., HSM).

Various subsystems within the studio interact with one another (for example, immersion production and metadata management). In one embodiment, each studio subsystem presents an application programming interface (API) for access by other subsystems. The process management 308 controls the automated movement of assets and metadata through the studio. It manages routes, process events, schedules, and rules, defines a common process management API that studio subsystems support and use this API to invoke particular asset and metadata operations in response to event triggers. The process management system may be tightly integrated with the monitoring and control system.

The result of the processes described above in conjunction with Figure 3 is the production of content. In one embodiment, the content may be web content. In one embodiment, a web content publishing system streamlines the web content production process. The web content publishing system may support file locking to prevent simultaneous updates by multiple users, version management, HTML link maintenance, specialized content verification triggers, incremental update generation, multiple staging areas, and automated content pushes. Web sites may have special needs for content publishing and delivery. First, the web page may need a graceful mechanism for dynamically updating the files being delivered by the site. Secondly, the web pages may need a robust, scalable infrastructure for delivering dynamic content (particularly content that is truly interactive, such as a multi-user on-line game). In one embodiment, the web content delivery system includes middleware and application software necessary to support these requirements. The web content delivery system may be a third-party different than the content generator.

In one embodiment, production 103 is able to create and distribute content in the form of incremental web site updates for ensuring incremental updates to live sites. To

perform incremental updates to the live sites, two versions of the data are maintained on one server. Each version is accessed through separate directories in the file system. While one version is being accessed by a server, the other version may be updated by a replication module that is capable of updating either file directory. When updates for all the storage locations have been completed, the directories are switched so that the server accesses the updated directory and allows the previously used directory to be updated. The directories need not be moved to implement the switch. In alternative embodiments, only the pointer used by the server to access a directory is changed. This ensures that the newest version is always available. In one embodiment, a version number is associated with each version to indicate which version is currently being stored. In such a case, latest version available on all servers is the version that is used and made accessible.

In one embodiment, production 103 uses hardware such as specialized archive equipment (tape backup systems, video servers, etc.), production management servers, video encoders, and network equipment.

Distribution

Referring back to Figures 1 and 2, distribution 104 is the process of transmitting streams and packages from the studio, via high-speed IP networks, to delivery facilities and/or mechanisms. Distribution 104 may use a distribution network having multiple simultaneously transmitting channels. In one embodiment, broadband communications are used for transmission.

In one embodiment, the mechanism(s) used for distribution 104 may be selected based on the ability to accommodate bandwidth management, reliability, and/or other considerations.

Delivery

Delivery 105 makes immersion content available to immersion 106. Numerous, different classes of delivery technology may be used, and multiple instances of each class may be used as well. In one embodiment, delivery 105 may employ satellite, cable, telcos, ISPs, television, radio, on-line and print.

Depending on the particular technology, delivery may take the form of one-way broadcast or client-server requests, via low, medium, and high-bandwidth bi-

directional networks. Also depending on the technology, delivery may or may not involve translating streams and packages into other formats (e.g., extracting data from a telemetry stream and inserting it into a relational database). Each delivery provider may implement a proprietary content reception protocol on top of basic IP protocols.

Depending on the combination of delivery and immersion technology, AI usage and user identification information generation may occur with delivery 105. AI information is discussed in more detail below. In the case where assets are encrypted before or during distribution, decryption may occur during delivery 105.

Immersion

Immersion 106 is the process of compositing immersion content for access by end users. Many immersion applications may be executed, or otherwise performed, in immersion 106. For example, immersion applications may include custom, standalone applications, generic HTML browsers and Java applets, etc. In certain cases, where end user devices are essentially non-interactive (e.g., traditional television), immersion 106 may occur as part of delivery 105.

Immersion 106 may comprise software that generates a generic web browser or a custom immersion platform. Such an immersion platform enables the end user to specify content desired for viewing, such as by selecting active links (e.g., Universal Resource Locators (URLs), thumbnails, etc.).

The immersion applications and software may be part of a client executing on an end user device. In one embodiment, each client (for use by the end user) is comprised of filter, an agent, and some type of display interface. The filter receives the stream from the delivery provider and filters out content to retain only that content that may be displayed by the client to the end user. The filter may understand the capabilities of the client, and based on knowledge of what data is being sent, can determine which content to filter (e.g., ignoring a stream of graphics or video in the use of a PDA type device). The filtering uses the content map or other control stream information to determine what is being sent and received and what information to ignore. Once filtered, the data is sent to the agent which coordinates with the display interface to display or otherwise provide the information to the end user. The agent is a client-specific agent designed to coordinate the display of information in a predetermined

way. The display interface interfaces the agent to the display functionality of the end user's device or system.

Monitoring and Control

Monitoring and control 107 monitors all segments of the process, with the exception of immersion and sometimes delivery, to control the systems within processes. The control may include bringing a system on and off-line as well as coordinating dynamic and static configurations. Monitoring and control 107 also ensures that the processes are synchronized with a global network time. In one embodiment, monitoring and control 107 is performed by a system running a system management application used for monitoring and controlling hardware and software systems throughout the platform in a manner well-known in the art. In one embodiment, the systems responsibilities include detecting hardware, software network failures, maintain hardware and software asset inventories, automatically configuring hardware based on software asset inventories, and notifying operators of failures or conditions. The system performs by using status information provided by the elements of the platform. In one embodiment, this information is provided on demand. In one embodiment, the monitoring and control system sends programmatic configuration commands to the elements (e.g., software systems and processes) of the platform to obtain the status information. An API may facilitate the defining of queries and commands to which the components in the system respond.

In one embodiment, in addition to monitoring and controlling individual resources, the system is able to monitor content streams and packages as they flow between physical resources. The system is able to determine such things as: where a given asset is, what path it is following, and why an asset failed to make it from point A to point B, where A and B may not be directly connected to each other.

Communication throughout the Platform

The platform may be implemented using well-known technologies, including, but not limited to, networks, hardware, transport protocols, and software. Transport protocols transfer the streams and packages. In one embodiment, all assets are transported over IP, with stream delivery using Real Transfer Protocol (RTP) (IETF RFC

1889) on top of IP and package delivery using a File Transfer Protocol (FTP) (IETF RFC 959) over IP.

In one embodiment, HTML (HTML 4.0 W3C recommendation) and XML (XML 1.0 W3C recommendation) are used to organize and present non-streaming content. In the case of satellite data broadcasting, for example, there is no backchannel and, thus, no opportunity for client-server HTTP sessions. File-based content is, therefore, broadcast along with streams, and cached on the client. The format of the cached content is still HTML. Included with the client (end user) software is an HTTP proxy that has access to the local cache.

HTML files may contain both content and links and are stored within the studio as assets. Their link structure is also accessible to the metadata management process. In one embodiment, each web site has a special "root" context. By navigating downward from the root, it is possible to reconstitute a web site from metadata and stored assets.

In one embodiment, audio/video data may not be transported in IP format. For example, there may be a particularly remote event where the only way to transfer video from the venue is via traditional television broadcast technologies. Similarly, a television broadcast station may be a delivery provider.

In one embodiment, each platform process comprises an individual IP network, which together form an internet. The end-to-end platform requires an internetwork that connects the segments into a seamless whole. A package delivery system provides a uniform mechanism for transporting packages between platform segments. A stream delivery system provides a uniform mechanism for transporting streams between architecture segments. In one embodiment, the media database comprises a relational or object-relational database that contains all metadata and supports all metadata management functions; however, any type of database structure may be used. It will be required to provide standard database functions, such as transactional semantics, recovery, hot backup, SQL queries, query optimization, and so forth. In one embodiment, the media database directly imports and exports XML documents.

The platform uses other various software components. In one embodiment, these components include systems (i.e., databases), middleware, applications, and API's. In one embodiment, applications use open internet standards (i.e., LDAP, FTP, SSL, XML, etc.), management GUI's use Web technologies, so as to provide location-independent

access, and API's are defined and implemented using the Object Management Group's CORBA standard CORBA/IIOP 2.2.

Context Management

The platform manages media assets, such as video, images, audio, and the like, using metadata. Context and other metadata are information that gives the assets meaning. Metadata is information about an asset, such as, for example, its type (e.g., JPEG, MPEG, etc.), its author, its physical attributes (e.g., IP multicast addresses, storage locations, compression schemes, file formats, bitrates, etc.), its relationship to other assets (e.g., that a photograph was captured from a given frame of a particular video), the situation in which an end user accessed it (e.g., a hit), its heritage (e.g., other assets from which it was generated), or its movement through the processing.

Metadata may be data that describes a particular image. For example, the knowledge that a particular photograph was taken at the German Grand Prix motorcycle race on a particular date, is stored in JPEG format, was used in the book Faster, Faster, Faster, and is of Mick Doohan on his Honda is metadata.

Metadata can also provide more abstract information such as, for example, the types of values available within a particular kind of telemetry stream, instructions generated by production and followed by immersion (e.g., to track certain kinds of end user behavior, to automatically present certain assets to the end user at certain times or in response to certain end user actions, etc.), or relationships between assets and other entities such as events, competitors, sponsors, etc.

As described above, in one embodiment, the platform treats both assets and metadata as first-class objects. Metadata can apply to other metadata. The platform transmits metadata from one location to another. For example, information about the syndicate that owns a boat can be distributed along with a photo of the boat.

Metadata may be used to provide information on what to do with assets. For example, metadata may be used to allow assets to be processed into a narrative (for example, by indicating what an asset is thereby triggering its inclusion into a narrative being produced), to coordinate their flow through a production process (for example, by indicting that assets are approved for public release when their release requires such approval), to store and retrieve them efficiently, and/or to control licensing (for

example, where an asset may only be published in a particular location (e.g., country)) and security.

A context is a special kind of metadata that defines a set of assets (and/or other contexts). Examples of some contexts include the telemetry from Alex Zinardi's crash at Long Beach, all video assets from the German Gran Prix, the winner of the 1998-99 Around Alone race, the general manager of each syndicate participating in the 1997-98 Whitbread race. A context may be understood as a query executed against the metadata known to the system. A context can be dynamically generated, or it can be persistently stored for optimal access. Persistent contexts, like other kinds of abstract metadata, are themselves assets.

Using contexts, metadata may be managed. In one embodiment, a context map is created as part of a pre-production task that defines the structure of an event and determines how assets gathered at the event will be organized. In one embodiment, the context map comprises a graph that indicates how various contexts are related.

The context map relates assets and metadata, as well as web sites, within hierarchies. Various hierarchical processes (processes using hierarchical descriptions) include creation, querying (e.g., find the web site rooted at "X"), editing (e.g., add, remove, and replace nodes), transportation (e.g., compress into and decompress out of the format). For example, applying incremental updates to web sites that have been distributed to ISP's uses a hierarchical description. Transporting context maps between the collection and production processes and describing differences between versions of asset or context hierarchies within the studio use hierarchical descriptions. These transmissions are also in the single format used to transport digital media assets. Note that the context map (or a portion thereof) is sent from production to collection to organize the assets being collected. The context map at the event venue may be expanded if data being collected is not in the context map.

In one embodiment, the format used for sending context maps lends itself to describing hierarchies because assets are described in terms of their parents and children. In one embodiment, the format is a complete, self-contained representation of a given hierarchy. In an alternative embodiment, the format is a set of changes (add, remove, replace) to be applied to an existing hierarchy.

The platform operates as a distributed system. Related assets and metadata may flow through that system in different ways and at different times. However, they are

correlated with each other using contexts. For example, consider the following hypothetical example:

Mick Doohan is a competitor in Grand Prix motorcycle racing. He competes in a particular Grand Prix race in Germany on a particular date. Video and telemetry streams from his bike during the race are collected and transmitted to the studio. The question is, how does the studio know where the incoming streams came from, or what they're about? Even if they somehow knew that the streams came from "Doohan's bike", does "Doohan" mean the same thing to the studio that it does to the remote production facility at the race?

There is a global naming scheme, and a mechanism to ensure assets can be reliably associated with the correct global names. Given the physically and logically distributed nature of the platform, this mechanism functions properly in a decentralized environment. In other words, it is possible to generate a global name for "Doohan's bike" either within the studio or within the remote production facility, and to propagate that name in either direction between the two.

Furthermore, it is possible to asynchronously associate attributes with newly created names. In other words, the mere presence of the global name "Doohan's bike" doesn't provide any further information, such as the fact that it's a Honda, it's sponsored by Repsol, and so forth. Contextual information of this sort may well propagate between collection and production processes separately from the name and the assets, and it may arrive either sooner or later. A global name is generated for the arrival of an asset, or for the creation of contextual information. Both collection and production processes are able to associate either an asset or a context with an existing name.

Figure 5 illustrates an example of a context defined using metadata. Referring to Figure 5, each node in this tree is a context. The asset *picture.jpg* is directly placed in two contexts:

1. Motor Sports : GP Motorcycle Racing : Events : German GP : Assets
2. Motor Sports : GP Motorcycle Racing : Competitors : Mick Doohan

The context "Mick Doohan" is also connected to the sponsor "Honda." By following chains in the tree, more information about a particular context (e.g., picture.jpg.) may be obtained. Contexts are designed to handle the basic asset management needs in a simple and intuitive way. Not every piece of information that

needs to be tracked for an asset is a context. Some are simply attributes. For example, "Former Grand Prix Winner" might be an attribute of Mick Doohan. If, in the future, it becomes important to be able to very quickly find all Former Grand Prix Winners, a context may be created for it.

In one embodiment, the platform accomplishes name mapping through the use of Uniform Resource Names (URNs), such as set forth, for example, in IETF RFC 2141. A URN is a persistent, location-independent, globally unique identifier. Every object, whether an asset or a context such as a competitor, team, or event, has a URN. Contexts refer to assets and other contexts via a URN. URN's may safely passed between platform segments. In one embodiment, every asset contain its URN embedded within it. In the case of packages, the URN may be represented by the package's file name; in the case of streams, the URN is included in a packet which is wrapped around each data packet. Alternatively, other global identifiers may be used (e.g., UID).

The use of URN's and URN references allows assets and metadata to safely be split apart and recombined. The following scenario illustrates the manner in which this process occurs. This scenario is based on the context illustrated in Figure 6. Referring to Figure 6, contexts in bold are created in the studio, while contexts in italics are created in the remote production facility (RPF). Dotted lines represent references by URN across context creation boundaries.

Thus, the studio creates a context for the German Grand Prix race, and for Mick Doohan as a GP competitor, and generates URN's for both. The RPF configures itself with the event and competitor URN's that it receives from the studio. To the event context, it adds the fact that Doohan (i.e., his URN) is a competitor in this particular event. In other words, to the list of competitors in the event content, the URN is added. The RPF captures a video asset from Doohan's bike and creates an URN for it. By basing the asset URN on the event URN it received from the studio, the RPF ensures that there will be no conflicts with URN's created at other events. The RPF transmits the video asset to the studio as a stream with the asset's URN embedded within it. The RPF separately generates a metadata message describing the complete event context and transmits that asset to the studio. This metadata asset includes the Doohan URN as a member of the event's competitor set and the video asset as a member of the event's assets set. The video asset contains a reference to the Doohan URN.

When the video asset arrives at the studio, the studio registers it by its URN. When the metadata asset arrives, the studio updates its internal version of the German GP event context (again, using the shared event URN to match the two). If the video arrives first, it will be queryable only by URN. Once the metadata asset arrives, however, a query for "onboard video from Mick Doohan's bike at the German Grand Prix" will now succeed.

The following is one embodiment of a definition of End-To-End Context, described as nine separate steps.

First, creation and management (insert, update, delete, etc.) of a hierarchical database of "contexts". Contexts define points of focus for event coverage. Points of focus are both static and dynamic. Static contexts include creation contexts and structural contexts. Creation contexts define systems from which assets are created (e.g., car #1 telemetry, trackside camera #2, video production suite #3, etc.). Structural contexts define the logical structure of the event (e.g., competitors, teams, tracks, etc.). Dynamic contexts define points of interest that occur during the course of a particular event (e.g., crashes, winners, red flags, etc.)

Second, tagging of assets with unique global identifiers reflecting their creation contexts.

Third, use of asset identifiers to associate assets with structural contexts (for example, indicating that car #1 telemetry is associated with competitor X), resulting in a complete coverage map for an event referred to herein as the "context map".

Fourth, serialization of context maps into an industry-standard interchange format (e.g., into a bitstream).

Fifth, transmission of serialized context maps between multiple production facilities for the purpose of synchronizing one or more distributed context databases.

Sixth, selection of context map subsets, referred to herein as "context views" or asset group, representing particular perspectives on an event (e.g., a crash consisting of two video streams from two different angles, along with the telemetry from each car involved in the crash, for a particular period of time). The selection of a view may be made by a producer. In one embodiment, three parts are used to generate a view: 1) an indication of the logical arrangement of the view that is to be shown, 2) a style sheet indicating how the content is to be presented visually, and 3) code (e.g., a browser or applet) to put the view on the screen.

Seventh, serialization of context views into an industry-standard interchange format.

Eighth, transmission of serialized context views between a production facility and delivery head ends for the purpose of enhancing users' ability to navigate and understand the assets they are receiving, as well as allowing producers to point out particular, interesting perspectives on the event. Different types of applications, connected to different kinds of content distribution channels, can generate different interactive presentations of a single set of context views, particularly where the end user devices/systems are only capable of presenting certain context views.

Ninth, the use of context maps to translate raw usage logs into meaningful business intelligence (e.g., understanding that a particular set of "click streams" means that end users are interested in a given competitor as opposed to some other competitor). For example, using a web log regarding a particular asset, the information and metadata associated with the asset, via the context map, may provide beneficial knowledge. For example, if end users often view a particular portion of content, a determination may be made as to what is in the content and such information may be important to, for example, a manufacturer or sponsor if they know their product or competitor is getting exposure.

Audience Information

Contextual information makes it possible to generate audience intelligence (AI) instructions to track every access to a digital media asset, such as, for example, an asset associated with a Honda motorcycle in the example above.

The ability to generate and analyze AI, to transport assets and metadata between platform segments, and to generate dynamic content depend on an understanding of what an event is. Creating such a taxonomy involves defining the relationships between such elements as teams, competitors, manufacturers, sponsors, governing bodies, etc.

Note that in one embodiment, both contexts and AI are represented using the same format as the digital media assets.

As discussed above, AI is a collection and analysis of meaningful information about any end user activity. The context provides the framework for what is considered to meaningful. The platform described above supports AI through the

following processes which are separated across multiple platform segments. First, AI is supported through the generation of production instructions instructing immersion applications as to what behavior to track. Second, AI is supported by immersion (client) applications generating usage information based on production instructions. Thirdly, AI is supported through the collection of usage information back from immersion applications and the collection of end user identification (e.g., demographics, etc.) information back from immersion applications. AI may also be supported through cross-analysis of the collected usage and end user identification information.

Large amounts of AI end user activity and identification information is transferred back to the studio from immersion 106 (via client software) and from servers at delivery 105. This data is stored in a AI data warehouse. In one embodiment, the warehouse is a standard data warehouse product that supports multi-dimensional queries. It will be used to support AI analysis activities. The acquisition and dissemination process implement the entry and exit points to and from the studio of AI information and instructions, respectively. The acquisition process is responsible for decomposing formatted information into assets and metadata, and routing it into the studio. In one embodiment, the dissemination process is responsible for composing assets and metadata back into the format for distribution out of the studio. AI data can be communicated via Internet, bulk file transfer (e.g., floppy) (where the acquisition process does not require a direct (Internet) connection between the client (end user) and the studio), or gleaned from prompts to end user for action (e.g., "Call us for a free T-Shirt").

An Exemplary Format

In one embodiment, the platform uses a universal format for transporting and storing digital media assets. The use of the format ensures interoperability between platform segments as well as the ability to retain asset associations during storage and retrieval. In one embodiment, the format is based on eXtensible Mark-up Language (XML) (XML 1.0 W3L Recommendation) and also defines the low level format for certain kinds of data streams (e.g., card telemetry, etc.), defines packaging formats for transporting and storing groups of assets, and maintains a dense message format for transporting and storing metadata, specifies format requirements for all asset types

(e.g., video must be MPEG 2, etc.), and specifies protocol requirements for underlying network transport layers (e.g., real-time streams must be transported using RTP over multicache, etc.). In one embodiment, content is recorded in the XML high-level layer, followed by the lower level definitions.

The platform uses a standard file naming process. This process can be either manual or automatic. In one embodiment, a client API provides a common set of format services across client platforms. In an alternative embodiment, in which all assets are stored in databases, there will be no asset files and therefore no naming convention.

In one embodiment, the precise meaning of asset file names depends on the nature of the asset. An asset may be either "static" or "dynamic". A dynamic asset is one that was created relative to a particular place and time during the course of an event (e.g., a daily update report, or a photograph of the finish of a race, etc.). A static asset is one that plays a relatively stable role in the overall structure of a presentation (e.g., a navigational button, or an index page, etc.)

Static assets can be of several types. Examples of these types include, but are not limited to, structure (index pages and the like), behavior (scripts and applets that control interactive behavior), metadata (descriptions of schemas, contexts, production instructions, etc.), presentation (purely visual elements), content (static content such as bios, rules, and so forth).

In the case of static assets, the context code field corresponds to a Context ID (CID) representing the asset's place within the presentation structure (e.g., "the gallery," or "the main section," "global navigational elements," etc.). In the case of dynamic assets, this field corresponds to the CID for the context in which the asset was created. In one embodiment, the CID is a six character code for the context.

A creation context defines a specific environment in which an asset was produced. In other words, the context code field of an asset's file name indicates the location from which the asset came. The creation context may or may not directly correspond to the context in which the asset is deployed as part of immersion production. Examples of creation contexts include, but are not limited to, a competitor's vehicle on a track during a particular session of a particular race, a

journalist's laptop on a mountainside over the course of a trek, the daily news update production process within the studio, the "medals by country" leaderboard for a particular Olympic Games.

Ownership and management of CID's is the responsibility of event producers. Producers may manage CID's in any way they like (i.e., manually, or through some sort of database application). They may distribute identifier lists (of CID's) to asset producers manually (i.e., on a sheet of paper), or via a web page, or with any other convenient mechanism. All the naming standard requires is that CID's be unique within an event, and that event and asset producers agree on their meaning.

In one embodiment, file names consist of a body, followed by a period, followed by an extension, which are 1-5 characters in length. It should take the natural form for a given asset type (e.g., HTML files should have an extension of ".html"). The format for bodies is specified below. In one embodiment, file names are case-insensitive.

While not requiring it, the file naming standard encourages the use of metafiles. In one embodiment, where metafiles are used, they adhere to the following requirements: the metafile uses the same name as the asset it describes, except with an extension of ".qmf"; the asset and metafile are transported together inside a ZIP file; this file uses the same name as the asset, except with the extension of ".qpg".

It is tempting to try to use extensions to encode asset types (i.e., ".mpg" for video, ".jpeg" for photos, etc.). Unfortunately, file formats and asset types don't necessarily map one-to-one. For example, a video asset could be either MPEG or AVI. On the other hand, both free-form text and email could be represented in a ".txt" file. For this reason, file extensions are only to used help operating systems launch appropriate authoring tools. High-level asset types are encoded within file name bodies.

In one embodiment, a filename body for a dynamic asset may consist of the following fields in the following order: event, data, context, sequence number, type.

In one embodiment, filename body for a static asset may consist of the following fields in the following order: event, context, name, type.

In the case where an asset really comprises multiple sub-assets, the leading fields define the asset, and the final field (type) defines the sub-asset. Sub-assets are known as asset variants. In one embodiment, where not specified more narrowly, the legal character set for a field consist of: A-Z, a-z, 0-9, _ (underscore).

In one embodiment, three characters represent the event (or other top-level context, such as network, studio, corporate) to which the asset belongs. A global list of event codes is used. Eight characters represent the date on which the asset was originally collected, in the following format – YYYYMMDD. This date is in GMT format. Six characters representing the CID to which the asset belongs. In addition, an event-specific list of contexts may be selected for each event.

In one embodiment, CID's starting with a numeric digit (0-9) are reserved for system-generated assets (e.g., replication system temp files, etc.). Numeric CID's are not used for static assets where there is no way to distinguish the start of the context from the start of the date, and thus no way to determine that the asset is in fact "static".

For a static asset, one to twelve characters uniquely identify the asset within its context.

For a dynamic asset, five numeric characters represent an asset's sequence of creation within a given event/date/context combination. This sequence monotonically increases. Depending on the event and context, the sequence may encode different kinds of information (such as time in seconds from midnight, time in hours:minutes:seconds, etc.). This standard neither specifies nor constrains any internal structure within sequence numbers.

Three characters represent the asset's type. This is a logical type, such as "thumbnail", "update email", "story headline", rather than a physical file format. In some cases, however, the two may map directly to one another (i.e., photograph and JPEG). A global, event-independent list of legal types is used. In addition, a list of event-specific type codes are selected for each event.

In the case where a metafile describes a set of assets differing only in the type field, rather than a single asset, the metafile uses a reserved type code. A metafile that describes a single asset variant can still use that variant's type code.

Following is an example file name for a daily update page created for an event:

- AAR19981001DAILUP00001PAG.html

Following is an example file name for a navigational button created for the Gallery section of the an event:

AARGALLRYnextBTN.gif

Following are example file names for logos to be used across events:

- QUOGLOBALCompanyNameLGO.gif ; version to be used throughout a site
- QUOHOMEPGCompanyNameLGO.gif ; special version for the home page

Following are example file names for a daily update report that combines a set of asset variants:

- AAR19981001DAILUP00001HDL.txt ; headline
- AAR19981001DAILUP00001SUM.txt ; summary
- AAR19981001DAILUP00001BDY.txt ; body
- AAR19981001DAILUP00001PAG.html ; compiled page
- AAR19981001DAILUP00001___.qmf ; metafile for the asset group

AAR19981001DAILUP00001SUM.qmf ; metafile for the summary variant.

A Client Architecture for Delivering Immersion

A scalable system for presenting immersive sports experiences is described. In one embodiment, the system presents an immersive experience containing both live and static assets for any event on any delivery platform. The system scales to any number of assets, any bandwidth, and any platform capabilities.

The architecture presents the end user with choices of how and what to asset group. The architecture is aided by the very design of the assets. Asset groups define associations between assets. Asset groups are, in turn, bundled with stylesheets which describe how to present those assets.

Figure 7 illustrates one embodiment of the architecture structure for an end user client. In one embodiment, the application 700 is divided into three main areas of functionality: the hardware abstraction 701, the core 702, and the presentation modules 703.

The hardware abstraction 701 provides an interface to hardware in the system that transfers information to and from the client. The hardware abstraction layer also insulates the core 702 and the presentation modules 703 from specific implementations of video streams, audio streams, input streams, display functionality, and local storage facilities. Thus, one embodiment, the hardware abstraction 101 interfaces a video decoder, audio decoder, display driver, end user input, local storage, etc., from the core 702 and presentation modules 703. In one embodiment, there is an API defined for each hardware capability. This layer registers each hardware resource with the core 702 to make them available to presentation modules 703.

The core 702 coordinates the visual presentation, manages end user input, and coordinates the media streams. In itself, the core 702 does not actually draw anything on the screen; it manages the process by which the display is rendered and relies on presentation modules 703 to actually perform the rendering. The core 702 also embodies the client services that implement features such as the context map, and view management.

In one embodiment, presentation modules 703 are loaded dynamically either from the local hardware platform or from a data stream. They implement the actual visual display that the users see and interact with.

An asset group is an arbitrary collection of assets that define the content presented by a stylesheet. The collection of assets may include both streamed and browsed assets. A stylesheet is a description of how those assets are visualized, it specifies which presentation modules 703 are used, and what content they will display. Figure 8 illustrates asset groups, stylesheets and presentation modules. Referring to Figure 8, the universe of assets 800 are shown with an asset group 801 identified. From the universe of assets 800, asset group 801, which tells a particular story, is selected and applied to a style that defines a layout for a set of presentation modules 803 that use the assets on a specific delivery platform.

In one embodiment, stylesheets can be nested in much the same way that frames can be nested in HTML. Figure 9 illustrates one embodiment of a stylesheet describing the layout and presentation.

Technical Architecture

The architecture of the client relies on the characteristics of the context map. The context map provides a structure for all event content. Figure 10 illustrates a simple context map. This map is also used to give unique identifications to assets (either by URN or UID). All references to assets are by these unique names, so it makes sense to use a client-side context map to manage the assets since the client will also be referring to the assets by these unique names. This client-side context map allows for a new mechanism for state management in presentation modules 703. Moreover, these names provide a consistent namespace for defining stylesheets and asset groups that apply to particular clients.

In one embodiment, a leaderboard component posted "Car X Selected" events when the end user clicked on a specific car. A BioViewer presentation module, having previously subscribed to that event type, would receive the event and look up the specified car number in a local data structure to find out who is driving that car. One problem with this design is that the BioViewer presentation module could be used in

just about any sport viewing experience, except for the fact that it depends on the "Car X Selected" event which may or may not exist in another sport.

The client-side context map also allows for a mechanism for state management in presentation modules 703. Change notification is the process whereby presentation modules 703 are informed of changes in the context map.

Change notification resolves a problem of having events being too tightly coupled to the code of individual presentation modules, and not capable of providing a structure which could be remapped to different data layouts for different sports by moving the semantic dependencies out of the presentation modules and into the context map. To use the previous example, the BioViewer presentation module, as specified in a particular stylesheet, subscribes to change notification on the "Current Racer" node of the context map. Rather than coding the context map specifics into the module, this relationship is established by the stylesheet that caused the BioViewer presentation module to be instantiated. When the end user clicks on the leaderboard, it updates the "Current Racer" node. The BioViewer presentation module is then notified of the change. A different stylesheet could establish a BioViewer that tracked the "Hero Racer" or the "Racer in the Lead" rather than the "Current Racer", with no code changes required in the BioViewer presentation module itself.

In one embodiment, context map node information are described externally to the implementation of the module. In one embodiment, presentation modules 703 may never have "hardcoded" references to particular context map nodes; all such references are defined in a stylesheet, such that the presentation module 703 can be repurposed merely by changing the stylesheet.

Data / Control Flow

In one embodiment, data flows from the collection interface through the platform-specific components, into the context map. Control is initiated in the user interface (via the presentation modules 703), except for master control directives (also known as "Priorities") delivered via production. These directives allow the production facility (the studio) to provide appropriate idle activity, guide the end user to content they might otherwise miss, to change the behavior of specific components such as the

AI module, or to impose a particular asset group and stylesheet on the client so as to focus the user's attention on a specific happening in the sporting event.

Event Categories

The data and control flow into the client is structured in terms of events. Note that this is different from an event such as a sporting event. In this case event refers to "an object within the client code that represents a particular state update." There are three categories of events that arrive from sources outside the client itself. These three categories of events drive the system. User events represent low-level user input (mouse clicks or keyboard input, for instance). User events are handled by presentation modules 703, perhaps after some filtering by the core 702. Timer events are generated by an internal system clock and are used to synchronize animation and rendering. Timer events are handled by the context map, and result in updates to the "current time" entry of the context map. Data events are created when new data arrives from the network or other transport medium (for example, "video frame available" or "telemetry datum available"). All data events result in updates to the context map.

Note that in one embodiment presentation modules 703 do not directly receive timer or data events; presentation modules 703 rely on the context map to obtain data and notifications about data updates. The timer and data events are therefore "hidden" by the context map. Presentation modules 703 that wish to update themselves as time passes do so by requesting change notification from the "current time" node of the context map. (The core 702 does provide services for logging and replaying all incoming events, to facilitate testing, but this is not visible to presentation modules 703.)

Input Event Filtering

The core 702 provides an event filtering service for input (e.g., mouse input, keyboard input, etc.). For example, when the end user moves the mouse, global mouse move events are generated by the user-input-handling platform component. These global mouse move events are then received by the display manager, which maps them to a particular display area owned by a particular presentation module. That presentation module then receives a local mouse move event which is within its own display area's coordinate space. This enables presentation modules to be ignorant of

their location on the screen; they need only deal with input handling relative to their own portion of the screen.

Context Map Entries

In one embodiment, there are two categories of context map entries, data entries and viewer entries. Data entries consist of information about the event delivered from the venue. In general, data entries in the context map are largely the same in all the clients viewing a given event. Some examples include: spectator, the current time, statistics about each participant, the leaderboard, in sports which have a leaderboard, the status of the entire event.

Viewer entries consist of information about what this particular end user is currently viewing or focusing on. In general, viewer entries in the context map may be different for each user's client. Some examples include: the currently selected contestant (e.g., the contestant that most interests the user), the current view being watched by the end user, the volume (as set by the end user) of various audio streams.

Context Map Update

Data entries are updated either by the core 702 itself in response to incoming data events, or by internal components (presentation modules) which compute new context information in response to that data. For example, the leaderboard could be a calculated entry in the context map, updated by a component in response to individual timing events. Thus, not all data entries in the context map are necessarily delivered from the venue; some may be calculated locally on the client.

Viewer entries are updated by presentation modules 703 in response to end user input or directorial commands. Updates to viewer entries by some presentation modules 703 may, in turn, result in context map update notifications to other presentation modules 103.

Overall Client Control Flow

Figure 11 is a schematic of the control flow when the end user clicks (for example) on a racer whom they want to select. Referring to Figure 11, the process begins with the mouse input is received by the user-input-handling platform module

(processing block 1101). Next, the user input event is routed to the display manager (processing block 1102). Then, the display manager calculates which display area was hit and routes a display area event to that presentation module (processing block 1103). Afterwards, that presentation module calculates which racer was selected, and updates a selected racer viewer entry in the context map (processing block 1104). The other presentation modules (e.g., the BioViewer presentation module and a telemetry viewer presentation module) receive update notifications from the selected racer viewer entry and refresh themselves accordingly (processing block 1105).

Interaction with Other Software

The client relies upon the client components to deliver an event stream. These components decode the information coming to the client and present interfaces for the various media types (e.g., audio, video, telemetry, control, etc.). Platform standard components handle the decoding of audio and video streams. To prevent the client from developing specific component dependencies, every widely available platform component are included with an abstraction layer that provides a standard interface to that functionality.

It is not necessary or cost-effective to try and abstract every external feature into a platform independent interface. In cases where a module has no possible use on in any other solution, it may be acceptable to eliminate the abstraction layer and bind the module directly to the external components.

Formats

Information is delivered to the application in either streams or packages formatted as described above. Packages may contain data files or browsed content to cached on the client. Specific formats may be supported with appropriate browser plug-ins.

The application may write content to the local storage facilities via predetermined format (e.g., XML). Where necessary, modules may use proprietary formats that are designed for specific use.

Major Modules

As described above with respect to Figure 7, the hardware modules present on one embodiment of a typically configured client are: video decoder 711, decoder 712, package receiver 713, telemetry 714, display 715, user input 716 and local storage 717. Video decoder 711 decompresses video streams into primary or secondary display surface. Audio decoder 712 decompresses audio streams into local audio playback facilities. Package receiver 713 handles storage and registration of locally stored content. Telemetry 714 decompresses telemetry streams into client APIs. Display 715 manages primary and secondary display surface allocation. User Input 716 handles local platform input devices. Local storage 717 provides storage and retrieval functionality for arbitrary data.

While the core 702 of the client is somewhat dynamic, the following comprise the functionalities for a context map, display manager, event dispatch, audio manager, and video manager.

The context map is local representation of the event content. The context map receives assets and stores them in a structured tree-like namespace. It provides access to particular entries by URNs. The context map also allows the creation and deletion of entries, the storage of structured data in individual nodes, and the notification of context map clients when nodes are updated or otherwise changed. Finally, there are two possible implementation strategies for handling the creation and maintenance of the context map itself. Either the context map object receives data events and self-manages its data and structure, or a separate data event handler receives the events and updates the context map through its API.

The display manager renders loop and handles animation clock, display loop, and display management. The display manager may be a simplified windowing system. In one embodiment, during initialization, a presentation module 703 allocates screen space as directed by the layout directives in the stylesheet that caused it to be loaded. This not only simplifies changes to screen layouts, it allows the display manger to quickly contextualize user input events. In addition to managing the display, this component also manages the render thread and the animation thread. These process threads are suspended and resumed via display manager interfaces. Every

presentation module 703 that requests a display area causes the associated drawing surface to be registered with the renderer. As a result, surfaces allocated as part of a display area can be double buffered if there are sufficient system resources. In the current implementation, presentation modules 703 assume that they are double buffered (calling flip() at the conclusion of drawing operations) in order to automatically take advantage of this functionality.

The event dispatch whose function is the coordination of event flow. The event dispatcher implements the following services: subscribe, cancel, and post. The subscribe (event description, handler) service registers an interest in a kind of event, returns a unique subscription-identifier. The cancel(subscription-id) relinquishes an interest in a previously subscribed event. The post(event) submits an event.

The event dispatcher is responsible for routing events. The event dispatcher has three jobs: it allows components to post new events; it allows components to subscribe to particular types of events; and it implements an internal dispatch loop which selects one event at a time and dispatches it to all the subscribers for that event's type.

In one embodiment, the event dispatcher supports three different techniques for posting events. The post external events method is used by external modules to post new, incoming, external events from the outside world. Post external events is thread-safe; multiple, separately-threaded external modules may all be posting external events concurrently and the event dispatcher will not block. The event dispatcher may reorder or queue various categories of external events before dispatching those events to subscribers. The post replay events method is very much like post external events, but is intended specifically for supporting replay of a recorded external event trace. Events posted with post replay events are guaranteed never to get reordered within the event dispatcher before being dispatched. This is critical for deterministic replay. The post internal events method is used by internal components to post viewer events which should not be recorded as part of the external event stream. In one embodiment, internal events are never reordered.

The event dispatcher's dispatch loop is essentially single-threaded; there is only ever one event being dispatched at a time, and when an event is selected for dispatch, it gets dispatched to each subscriber in order, one at a time. So the guarantee is that when an event subscriber gets called with an event, it is assumed that no other component in

the system is currently handling that event. This calls into other modules in the system, safe in the knowledge that those modules will not be busy, and hence no deadlock.

The audio manager handles balance, volume, and stream allocation for audio. The audio manager is an API used by the presentation modules 703 to control an audio hardware module.

The video manager handles stream allocation, and destination assignments for video. The video manager is an API used by the presentation modules 703 to control a video hardware module.

The list of presentation modules 703 in one embodiment of a client map consists of:

- Initialization - read local configuration information, instantiate rest of presentation modules
- Video thumbnail viewer - present video source options
- Primary video viewer - present currently selected video feed
- Dashboard - visualize telemetry from a specific car
- LapSplitViewer - visualization of the lap time differences between top cars
- Biography - display biographical information for a specific driver or car
- Leader Board - scrolling seismograph of race, current positions of each car
- Chronicle - access to cached assets
- TrackView - a 2d view of the track
- Event Log - telemetry storage, access by time-frame / car / etc.

Figure 12 illustrates the application as a collaboration between many different components, each supplying a specific area of functionality.

In addition to the specific modules of functionality, the following API's that include a presentation module API and common hardware distribution APIs may be

used. The presentation module API specifies the interfaces to the core functionality that is available to the presentation modules. The common hardware abstractions include specific API's for each of the hardware components.

The architecture described herein may be built in any environment that allows component blocks of code to be loaded on demand. Java may be used to implant the client but this does not preclude other solutions such as C++/Visual Basic with COM or ActiveX, shared libraries, or even TCL/TK.

Event flow

Corresponding to the of modules above, there is a straightforward event flow within the system, which drives the entire client.

External events are events which arrive from outside the client; they represent the outside world's entire effect on the client. Input events, data events, participant events, and time events make up all the external events.

Internal events are generated by modules of the client in response to particular external events. These internal events include (for example) leaderboard update, racer selection, etc.

The client is fundamentally based on the concept of deterministic execution. That is, everything that happens in the client happens in response to handling external events. The external events received by the client can be recorded in linear order, and if the client is later restarted and those exact events are replayed into the client, the client will do exactly what it did on the previous run of the system.

An Exemplary Immersion Presentation

One form of the content that may be made part of immersion production for an event is a chronicle of the event. In one embodiment, the chronicle of the event is only slightly less than real time. In such a case, the occurrences of an event may be depicted in minutes rather than hours. The system may be used for live sportscasting, and thus the system produces live coverage of a "chronicle-like" nature. In one embodiment, the term "live" when used in conjunction with a "chronicle" refers to being produced and published within a predetermined period of time (e.g., five minutes). By "chronicle-like," the coverage comprises a series of produced self-contained "stories," each of

which may have its own complex multimedia layout. For instance, chronicle content being presented may include streamed coverage such as live video, real time audio coverage and live telemetry displays in a complementary way.

To create a chronicle, the chronicle is initially "sketched out" in pre-production. Using a rundown manager, the director begins to sketch out the structure of the broadcast. In one embodiment, the run down manager is an application that allows individuals to communicate with a rundown server, which acts as a backend for the system. The rundown itself is a list of POPs ("pieces of programming"). A POP is the basic unit of presentation, roughly corresponding to a "story" in broadcast news. At this stage, each POP is little more than a name. For example, a rundown for a Tour de France stage in early pre-production might begin something like the table below:

Rundown: Tour Stage 5 Live Coverage			
Sportscast date: 16 July 1998			
ID	Name	Comments	Template
1	Weather update	Use Cholet satellite image. Keep it to one HTML page, or two at most.	
2	Yellow Jersey profile	Make it personal - what he had for breakfast	
3	Contender profile 1	Let's build some rivalry interest with this if we can	
4	Contender profile 2	An American if possible	
5	Injury update	Use the same Template from last year	
6	Start	Maxine has ideas about how to format this - talk to her	

The early rundown provides a skeletal structure for the sportscast, but POPs will be added, deleted, modified, and re-ordered right up through the live presentation.

The next pre-production step is to start building templates. The templates define the structure of the POP - what kind of media is in it, and how it is laid out. The rundown defines the pacing and order of the event being broadcast; the templates define its look. In one embodiment, templates are HTML based, and are described in detail below.

In general, templates are created for specific coverage situations that can be anticipated, although some generic templates may also be useful. The production of a template is a two-stage process. In one embodiment, templates begin with actual HTML composed by a designer. In one embodiment, there is no limit to the size of a template; it may extend to many pages. Once composed, the design and media elements of the layout which are fixed and those which are variable are indicated. Appropriate script code is added to the HTML to produce a final template.

In production, the template is filled with content during the course of the live production process. When it is ready to be published, it is compiled back into pure HTML and handed off to a publishing system. This may be automated or manual.

As images and video come in from the event, they are processed (automatically or manually) and attached to the appropriate templates. As text and data come in from the field, they are also added to the templates. An individual controls the priority of the various POPs in progress. The rundown manager (a software program in one embodiment) keeps everyone on the team informed regarding the status of the POPs and what they need to do next.

A template captures the key elements of the design in such a way that the layout can be used with whatever particular content is required for live coverage. This moves the bulk of the design and layout work into pre-production, and makes live event broadcasting possible.

In one embodiment, the template for the page is created directly from the working HTML. The variable portions of the design are then separated from the static portions. Possible static portions might be corporate branding, or elements that are part of the presentation as a whole (e.g., background colors, font sizes, or even background images).

Once converted, the file is no longer valid HTML:

```
<HTML>
{{
define(name='Background', type='image')
define(name='Body', type='text')
define(name='Epilog', type='text')
define(name='Signature', type='text')
}}
<HEAD>
<TITLE>Untitled</TITLE>
</HEAD>
<BODY bgcolor="#000000" link="cccccc" vlink="cccccc" alink=ffffff
background="{{get('Background')}}">
<table width=950 height=435><tr><td VALIGN=center>
<font face="helvetica, ariel, arial" size=3 color=#ffffff>
{{get('Body')}}
</font></td></tr>
<tr><td valign=bottom align=right>
<font face="helvetica, ariel, arial" size=5 color=#cccccc>
{{get('Epilog')}}</font><br>
<font face="courier" size=5 color=#ffffff>{{get('Signature')}}</font></td></tr>
</table>
</BODY>
</HTML>
```

As shown above, the dynamic (or changeable) elements of the design have been identified and specified in an initial definitions block, and the corresponding locations in the HTML have been replaced with syntax for retrieving those defined values.

Text color fields may be added to the template to help with legibility:

```
{{
```

...

define(name='Body Color', type='HTML color', default='#ffffff')

define(name='Epilog Color', type='HTML color', default='#aaaaaa')

define(name='Signature Color' type='HTML color' default='#ffffff')

}}

Then in the appropriate HTML, the text colors would be referenced:

...

{{**get('Body')**}}

</td></tr>

<tr><td valign=bottom align=right>

{{**get('Epilog')**}}

{{**get('Signature')**}}</td></tr>

...

These refinements allow the template to flex to accommodate a wider variety of content while retaining the structure and composition that the designer intended.

The script sets for the template syntax. The purpose of this template syntax is to allow broad flexibility in describing how content gets incorporated into the production of an HTML file. In achieving this goal, it balances legibility, extensibility, and complexity. Thus, a scripting language attempts to capture the complexity of a design.

The interpretation of the script is divided into two phases: field definition and compilation. Field definition occurs when the template is loaded into the rundown manager. In one embodiment, the template is partially parsed and any define() or declare() statements are executed to produce a suitable user interface (UI) for the template. In one embodiment, these statements contain information that determines which user-interface elements are displayed; text entry boxes for text fields, sliders for integer values, color pickers for HTML colors, and so forth.

Compilation takes place when the template is converted from script to HTML in a process also referred to as 'compiling'. During this phase, the statements and expressions that are not part of define() statements are interpreted, references to external media are resolved, any implicit media conversions are performed, and pure HTML is produced.

In one embodiment, the first construct of the script is the define statement. It is used to describe a part of the template that will be exposed in the production tools user interface.

define(*specification-list*)

A *specification-list* is a variable length list of comma separated parameters. The contents of the list depend upon what kind of variable is being defined. All definitions require a name, and a type, and all variables can be marked as required or optional, but beyond that there is no expectation of commonality. The documentation for each variable type has all of the details for its specification.

As an example, some possible definitions of a text field called 'Body' are given below:

define(name='Body', type='text')

define(name='Body', type='text', optional)

define(name='Body', type='text', minLength='50w')

The first definition is probably the simplest possible; it describes an unconstrained field of text. The second one is declared as optional, which indicates that there is no problem if the field is not filled in when the template is compiled. Finally, the last one has a constraint, perhaps imposed by the designer, that the text must be at least fifty words long.

The declare statement is used to expose static elements of the original design to the script context.

declare(*specification-list*)

For example, if a portion of the design is dependent on the width of a logo, it may be convenient to declare the logo and then refer to it in expressions during compile time.

declare(name='companyname-logo', src='$repository/tdf-logo.gif')

This declares that there is an image called tdf-logo. Declarations do not appear in the production tools user interface since there is no input required.

Declarations and definitions can refer to previously defined declarations. In this way the system can be used as a kind of macro language. In one embodiment, blocks of declarations and definitions can be imported as well.

Compile time syntax is very similar. Statements refer to previously defined or declared variables and are evaluated to produce HTML. In one embodiment, every statement either generates no output, or HTML.

The most common statement is **get()**. This function retrieves the value of a variable. How that variable is represented in HTML depends entirely on its type and any processing directives specified when it was declared or defined. For example:

get('companyname-logo')

would result in something like "http://www.companyname.com/images/tdf-log.gif". It will be appreciated by those skilled in the art that there are plenty of possible system configuration settings that would affect how the address is compiled.

As a more complex example, suppose that a table is to be as wide as an image. For the sake of brevity only the relevant portions of script and HTML are shown.

define(name='Body', type='text')

define(name='Headshot', type='image')

declare(name='border', type='int', value='15')

...

<table width={{**width('Headshot')+get('border')**}} height=435>

...

Here is an example that defines a style that affects how each paragraph of a text field is displayed:

define(name='Body', type='text', paraStyle='table_cells')

declare(name='table_cells', type='paraStyle',
spec='<tr><td>$paragraph</td></tr>')

...

<table>{{**get('Body')**}}</table>

Since the text variable has a paragraph style specified in its definition, it generates table row entries for each paragraph according to a predetermined specification of the corresponding style when it is evaluated during compilation. This shows an example of compile-time processing of fields. This kind of processing can also be triggered when the template contains mutable data that is not in precisely the format specified in the definitions. For example, it is possible to put a color image into a field that is constrained to black and white since that is a conversion that can safely be handled during compilation. The source image is kept in its original format, and the image referred to in the compiled HTML will be created at compile time.

In one embodiment, to create a template, a text editor, each variable element is identified and defined or declared as appropriate. Their original references in the HTML are replaced with embedded script statements. In this manner, the file is converted piece by piece into a template.

Once a template is deemed complete, it is wrapped up. This final step prepares the template for use in the live production environment. In one embodiment, a thumbnail image of the template compiled with sample content is generated, the template is given a name, and the template itself is placed in the production template repository. Once it has been wrapped up, the template it can be used in live production.

Rundown Manager

From a technical perspective, there are two parts to the rundown manager: the rundown server and the rundown client. The rundown server maintains the rundown and the individual POPs and communicates with the clients, the publishing system and the content management system, both of which are part of production. It works closely with the content management sub-system to provide seamless propagation of digital media assets as it is edited and submitted.

The rundown client provides a user interface through which the team members can work with and get information about the POPs in progress.

In one embodiment, when a POP is created, it is given a name, an automatically assigned unique ID, and a media scratchpad. The name ID and scratchpad stay with

the POP from the moment of creation, through production and publication, and are retained when the POP is archived after the sportscast. Of course, they may change the name at any time.

Once a POP exists, it can be assigned a template. In one embodiment, as with other digital media assets, the template can be removed or changed at any time. Digital media assets can be placed in the template or in the scratchpad. The scratchpad is a place to put raw source materials, possible alternate digital media assets for the template, or digital media assets that might be used for additional coverage in the future.

Below is a complete list of the contents of one embodiment of a POP:

Primary contents	Name	brief description of POP
	ID	automatically assigned unique identifier
Secondary Contents	Status	field- and POP-level status
	Template	the currently assigned Template
	Scratchpad	place to put source materials that are not yet in the Template
	Priority	the relative importance of a given POP
Optional Contents	Assignment information	who is responsible for what
	Deadline	time by which the POP must be completed

In one embodiment, the rundown client tailors its display of the POP list to the role of the person using it. So, the exact representation of the list, or a POP entry, depends upon who is looking at it.

In one embodiment, once a template has been assigned, the display of the POP includes suitable input areas for the fields specified in the template. For example, text fields will be represented as text entry boxes, and HTML color fields will have a color palette selector.

In addition to providing information on the state of the sportscast, the rundown manager also allows individuals to perform operations on POPs and on the POP list. A UI may be used to carry out these operations.

Operations on the POP List

Create and Delete POPs: The POP entries are created in the rundown manager in pre-production and during the live event broadcasting.

All that is needed to create a new POP is its name. No other information is required for a POP. An initial template may be supplied or loaded into either the scratchpad or the template. He or she can also change the current template or remove media from the template.

Re-order POPs: Unpublished entries may be moved up and down the list as their relevance or timeliness dictates.

Operations on a Particular POP

Add or Change a Template: In one embodiment, to assign a template to a POP, an individual drags a template from the template browser and drops it on the POP. If a template has already been assigned, then a dialog is displayed confirming the replacement operation.

Add media to the Template: Digital media assets of any type may be dragged directly from an editor and dropped on an appropriate field in the template interface.

Add media to the scratchpad: Digital media assets of any type may be placed anywhere in the media scratchpad.

Preview the POP: In one embodiment, every time new content is placed in a POP, it is automatically rendered out to the browser window. This feature can be turned off in the rundown manager configuration dialog. There may also be a 'Preview' button in the POP interface for looking at the current content of a template without having to add anything new to it.

Submit changes: Changes to content in a template remain local until the individual chooses to share them by "submitting" them.

Set Priority: The production team may be focused to particular POPs by increasing their priority.

Task assignment: A particular template field or entire POP may be assigned to a specific individual. In this situation, the fields will be locked to everyone except the assigned person.

Publish: Once the required fields of a POP have been filled, it may be published. There may be circumstances where they elect to publish a partially filled POP, and this is handled by confirming the operation when the warning dialog is displayed.

In one embodiment, once published, a POP may not be deleted or re-ordered. New entries may be inserted into the chronology after a POP that has been published.

In one embodiment, one of the design goals of the rundown manager is provide a mechanism for integrating familiar off-the-shelf tools into the live production environment. Where possible, this integration is facilitated via the Windows cut-and-paste buffer or the drag-and-drop interface.

Exemplary custom tools include a template browser and task accelerators. The template browser provides a mechanism for selecting a template by name, by thumbnail, or semi-automatically. In semi-automatic operation, the template browser offers possible selections based on which templates have been published recently. This feature is designed to help prevent unwanted template repetition in the sportscast.

Task accelerators may be used with specific tasks that may be fully automated. For example, a template may specify that a certain image must be black and white. This is called a constraint on the field. It is possible to tie the constraint to an automated task, in this case a graphics filter that converts the image given to it to grayscale. When a color image is dropped on the field in the template interface, it would be automatically converted. If there is no task tied to the constraint, then the image would be rejected and it would be up to an individual to edit the image.

By the time of publishing, a great deal of contextual information will have been accumulated in the POP. This information has value because it provides high-level descriptions of what would otherwise be mostly anonymous content. Therefore, when a POP is published it is archived with all of its assigned content, its template, and its scratchpad. The act of publishing causes a final compile to be performed to create the

HTML and localize all of the media references. This data is then published through whatever mechanisms are in place for that live production. The published data is also archived with the POP.

Furthermore, when a POP is published, it can be tagged in various ways to identify it as a "highlight of the day (or quarter or leg)," or a candidate for the appropriate summary chronology.

Archived POPs can be viewed with the rundown manager. They can also be copied and edited or re-compiled for different output resolutions.

Thus, the exemplary chronicle content is designed to enable the production team to create portions of live programming very quickly. Once this can be done, there is another issue to tackle: how to deliver the material to the audience. The audience is able to both receive the presentations passively and navigate among them. This is what will make the experience most exciting and bring it closest to perfectly emulate actually being at the event. To deliver the best experience to the audience, the publishing system provides: dynamic updates as POPs are published, and support for active and passive viewing, there are many options for the implementation of these features. At the low end of the spectrum is a simple HTML design that is pushed out to the end users every time a POP is published, and at the other end is an applet or embedded application that has a connection back to the content server

In between there are hybrid solutions that are more or less browser independent, using exotic technologies such as JavaScript, LiveConnect, Scriptlets, and DHTML.

Any of the Web server push systems could be used to propagate POPs as they are published.

Applets, embedded applications, and plug-ins may be used. In these solutions, a custom interface module is developed that provides a dynamically updated display. To achieve this, the interface module opens a connection back to a server and listens to status messages. Icons, pictures, or other markers appear in the interface as POPs are published and the POP display is automatically updated.

Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and

described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as essential to the invention.

Thus, an architecture for controlling the flow and transformation of multimedia data has been described.

CLAIMS

We claim:

1. An architecture comprising:

a plurality of collection devices to capture data from a remotely occurring event;

a remote production unit coupled to the plurality of collection devices to convert the data into digital media assets, the remote production unit tagging each of the digital media assets with a global identifier indicative of a creation context associated with said each digital media asset;

at least one communication mechanism to transmit the digital media assets;

a production studio coupled to receive and process the digital media assets and to select at least one context view, the production studio serializing the context view into an interchange format; and

a distribution network coupled to the production studio to distribute the digital media assets related to selected context views as a serialized bit stream.

2. The architecture defined in Claim 1 wherein the digital media assets are transmitted from the plurality of collection devices to the distribution mechanism using an industry standard protocol.

3. The architecture defined in Claim 2 wherein the industry standard protocol comprises an IP protocol.

4. The architecture defined in Claim 1 wherein the remote production unit devices associate context information with each of the digital media assets as the digital media assets are being collected.

5. The architecture defined in Claim 1 wherein the global identifier comprises a Uniform Resource Name (URN).

6. The architecture defined in Claim 1 wherein the global identifier comprises a persistent location-independent, globally unique identifier.

7. The architecture defined in Claim 1 wherein the production studio sends a context map to the remote production unit.

8. A method of depicting an event comprising:

capturing data from a remotely occurring event;

converting convert the data into digital media assets, including tagging each of the digital media assets with a global identifier indicative of a creation context associated with said each digital media asset;

transmitting the digital media assets;

processing the digital media assets to select at least one context view, including serializing the context view into an interchange format; and

distributing the digital media assets related to selected context views as a serialized bit stream.

9. The method defined in Claim 8 wherein the digital media assets are transmitted from the plurality of collection devices to the distribution mechanism using an industry standard protocol.

10. The method defined in Claim 9 wherein the industry standard protocol comprises an IP protocol.

11. The method defined in Claim 8 further comprising associating context information with each of the digital media assets as the digital media assets are being collected.

12. The method defined in Claim 8 wherein the global identifier comprises a Uniform Resource Name (URN).

13. The method defined in Claim 8 wherein the global identifier comprises a persistent, location-independent, globally unique identifier.

14. The method defined in Claim 8 further comprising sends a context map to a remote production unit performing conversion of captured event data.

15. An apparatus of depicting an event comprising:

means for capturing data from a remotely occurring event;

means for converting convert the data into digital media assets, including tagging each of the digital media assets with a global identifier indicative of a creation context associated with said each digital media asset;

means for transmitting the digital media assets;

means for processing the digital media assets to select at least one context view, including serializing the context view into an interchange format; and

means for distributing the digital media assets related to selected context views as a serialized bit stream.

16. The apparatus defined in Claim 15 wherein the digital media assets are transmitted from the plurality of collection devices to the distribution mechanism using an industry standard protocol.

17. The apparatus defined in Claim 16 wherein the industry standard protocol comprises an IP protocol.

18. The apparatus defined in Claim 15 further comprising means for associating context information with each of the digital media assets as the digital media assets are being collected.

19. The apparatus defined in Claim 15 wherein the global identifier comprises a Uniform Resource Name (URN).

20. The apparatus defined in Claim 15 wherein the global identifier comprises a persistent, location-independent, globally unique identifier.

21. The apparatus defined in Claim 15 further comprising studio sends a context map to the remote production unit performing conversion of captured event data.

22. An article of manufacture having one or more recordable media for storing executable instructions thereon which, when executed by one or more processing devices, cause the one or more processing devices to:

capture data from a remotely occurring event;

convert the data into digital media assets, including tagging each of the digital media assets with a global identifier indicative of a creation context associated with said each digital media asset;

transmit the digital media assets;

process the digital media assets to select at least one context view, including serializing the context view into an interchange format; and

distribute the digital media assets related to selected context views as a serialized bit stream.

23. A method of broadcasting an event comprising:

capturing data of a remotely occurring event;

sending the captured data to a production unit;

converting data into digital media assets using the production unit;

sending the digital media assets to a studio using an IP protocol;

processing the digital media assets;

disseminating processed digital media assets to a distribution network using the IP protocol; and

distributing the digital media assets using a plurality of channels to end users using the IP protocol.

24. The method defined in Claim 23 further comprising associating context information with each of the digital media assets as the digital media assets are being collected.

25. The method defined in Claim 23 further comprising tagging each of the digital media assets with a global identifier.

26. The method defined in Claim 25 wherein the global identifier comprises a uniformed resource name (URN).

27. The method defined in Claim 25 wherein the global identifier comprises a persistent, location-independent global unique identifier.

28. An apparatus for broadcasting an event comprising:

means for capturing data from a remotely occurring event;

means for sending the captured data to a production unit;

means for converting data into digital media assets using the production unit;

means for sending the digital media assets to a studio using an IP protocol;

means for processing the digital media assets;

means for disseminating processed digital media assets to a distribution network using the IP protocol; and

means for distributing the digital media assets using a plurality of channels to end users using the IP protocol.

29. The apparatus defined in Claim 28 further comprising means for associating context information with each of the digital media assets as the digital media assets are being collected.

30. The apparatus defined in Claim 28 further comprising means for tagging each of the digital media assets with a global identifier.

31. The apparatus defined in Claim 30 wherein the global identifier comprises a uniformed resource name (URN).

32. The apparatus defined in Claim 30 wherein the global identifier comprises a persistent, location-independent global unique identifier.

33. A method for broadcasting a sporting event comprising:

collecting multiple inputs of captured information from the sporting event;

converting the multiple inputs of captured information into digital media assets;

processing the digital media assets into sporting event content; and

distributing the sporting event content to a plurality of end users via a plurality of delivery mechanisms in such a way as to enable virtual emulation of the experience of the sporting event to the end users.

34. The method defined in Claim 33 wherein at least one of the multiple inputs comprises video and audio of the sporting event.

35. The method defined in Claim 33 wherein at least one of the multiple inputs comprises video and audio of individual competitors in the sporting event.

36. The method defined in Claim 33 wherein at least one of the multiple inputs comprises a telemetry stream of at least one vital function of at least one competitor.

37. The method defined in Claim 36 wherein the at least one vital function comprises competitor biometrics.

38. The method defined in Claim 33 wherein at least one of the multiple inputs comprises a stream of commentary on the sporting event.

39. The method defined in Claim 33 wherein distributing the sporting event content to the plurality of end users comprises sending an amount of sporting event content that is greater than an individual end user is able to experience at one time to constrain the individual end user to direct attention to individual portions of the sporting event content.

40. An apparatus for broadcasting a sporting event comprising:

a plurality of collection devices to collect multiple inputs of captured information from the sporting event;

a production unit to convert the multiple inputs of captured information into digital media assets;

a studio to process the digital media assets into sporting event content; and

a distribution network having a plurality of distribution channels to distribute the sporting event content to a plurality of end users via a plurality of delivery

mechanisms in such a way as to enable virtual emulation of the experience of the sporting event to the end users.

41. A method for processing information from an event, the method comprising:

capturing multiple input streams of data at the event; and

converting the multiple input streams into digital media assets having a predetermined format using a context map to organize the digital media assets and indicate relationships between contexts associated with the digital media assets.

42. The method defined in Claim 41 further comprising:

distributing content for delivery to at least one end user; and

presenting the content to the at least one end user.

43. The method defined in Claim 41 wherein converting the digital media assets into content comprises tagging each of the digital media assets with a global identifier.

44. The method defined in Claim 43 wherein the global identifier comprises a Uniform Resource Names (URN).

45. The method defined in Claim 43 wherein the global identifier comprises mapping a persistent, location-independent, globally unique identifier to digital media asset.

46. The method defined in Claim 41 wherein converting the digital media assets into content comprises generating context information at collection and at production.

47. The method defined in Claim 41 wherein the digital media assets comprise metadata and individual units of digital content.

48. The method defined in Claim 41 wherein the digital media assets comprise first-class objects.

49. The method defined in Claim 41 wherein the event comprises a sporting event.

50. The method defined in Claim 41 further comprising:

collecting event data and converting the event data into the digital media assets.

51. The method defined in Claim 41 further comprising selectively capturing the event data to dynamically control selection of the digital media assets.

52. The method defined in Claim 41 wherein converting the digital media assets into content comprises parsing the digital media assets into contexts, end user feed back information and individual units of digital content.

53. A method for broadcasting an event comprising:

disseminating content for delivery, the content corresponding to a plurality of input streams of remotely captured event data converted and formatted into digital media assets, wherein disseminating content comprises sending content to a plurality of channels based on characteristics of the device designated to display the content;

distributing the content to a plurality of end users via a plurality of delivery mechanisms; and

filtering the content based on user input to display a selected portion of the content.

54. The method defined in Claim 53 wherein disseminating content comprises serializing context views and transmitting context views to the delivery mechanisms.

55. The method defined in Claim 53 further comprising:

generating a plurality of interactive presentations of a single set of context views.

56. The method defined in Claim 55 further comprising:

generating end user activity information.

57. The method defined in Claim 56 further comprising:

generating instructions instructing immersion applications on user behavior to track.

58. The method defined in Claim 57 further comprising the immersion applications generating usage information.

59. The method defined in Claim 56 wherein the end user activity information and the digital media assets are formatted identically.

60. The method defined in Claim 53 wherein filtering comprises an end user device determining whether its display mechanism is capable of displaying a portion of the content, and ignoring the portion of the content the display mechanism is not capable of displaying the content.

61. An apparatus for broadcasting an event comprising:

means for disseminating content for delivery, the content corresponding to a plurality of input streams of remotely captured event data converted and formatted into digital media assets, wherein disseminating content comprises sending content to a

plurality of channels based on characteristics of the device designated to display the content;

means for distributing the content to a plurality of end users via a plurality of delivery mechanisms; and

means for filtering the content based on user input to display a selected portion of the content.

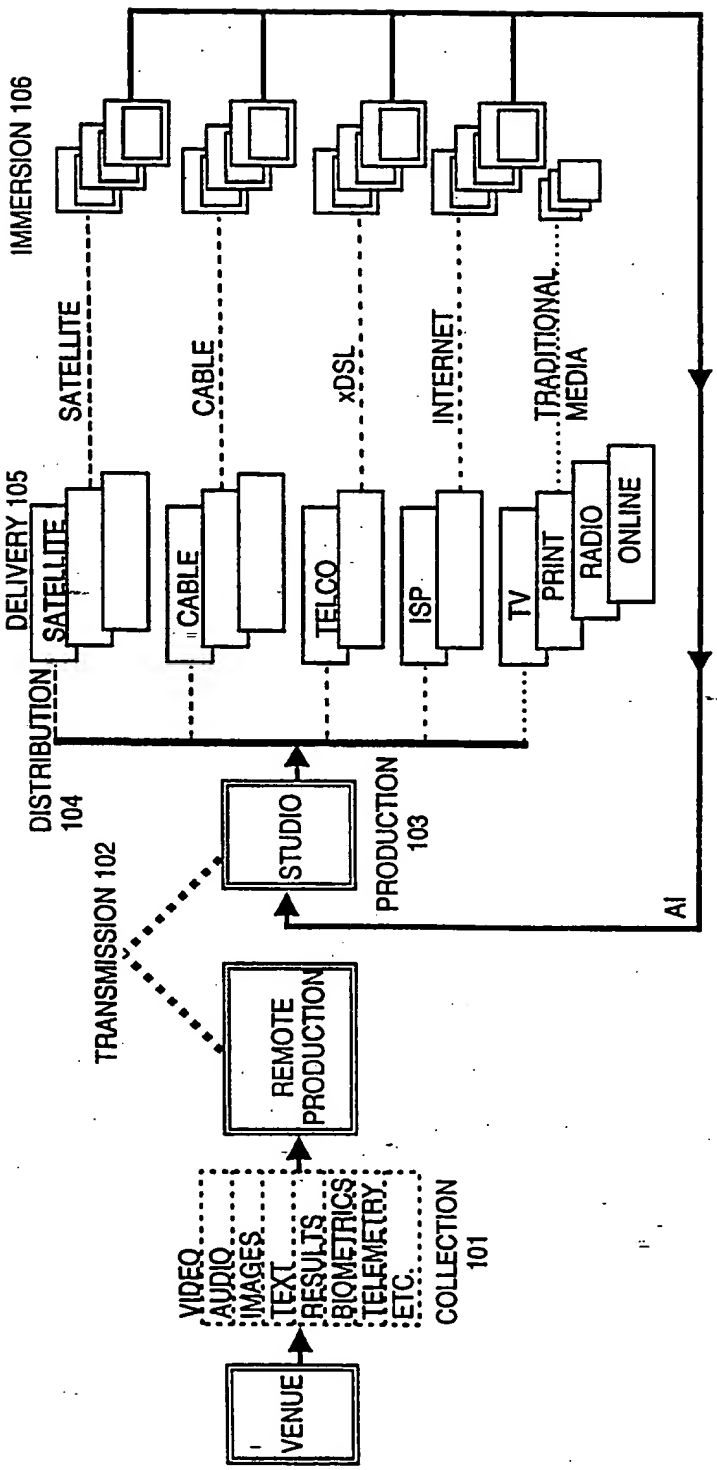
IMMERSION 106
SATELLITE
CABLE
xDSL
INTERNET
TRADITIONAL MEDIA
DELIVERY 105
SATELLITE
CABLE
TELCO
ISP
TV
PRINT
RADIO
ONLINE
DISTRIBUTION 104
PRODUCTION 103
STUDIO
AI
TRANSMISSION 102
REMOTE PRODUCTION
VIDEO
AUDIO
IMAGES
TEXT
RESULTS
BIOMETRICS
TELEMETRY
ETC.
COLLECTION 101
VENUE

FIGURE 1

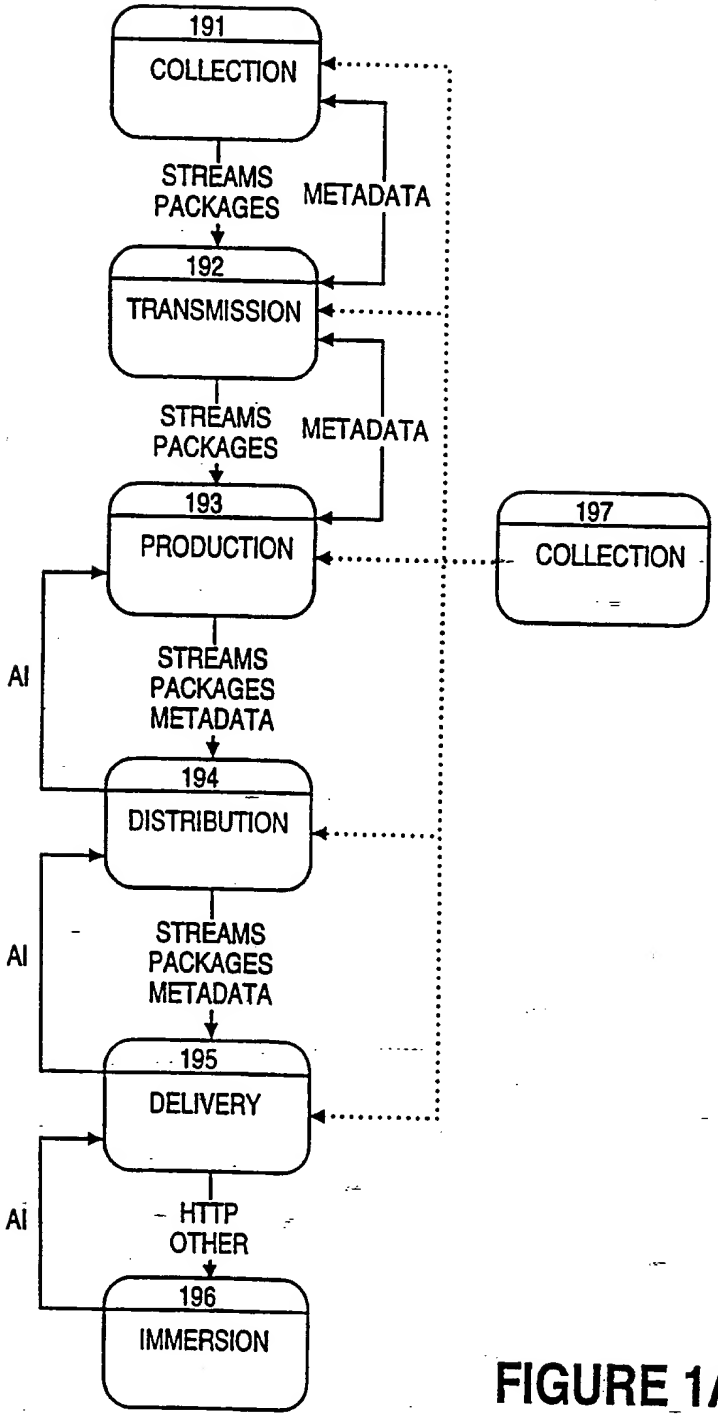
191
COLLECTION
STREAMS PACKAGES
METADATA
192
TRANSMISSION
STREAMS PACKAGES
METADATA
193
PRODUCTION
197
COLLECTION
AI
STREAMS PACKAGES METADATA
194
DISTRIBUTION
AI
STREAMS PACKAGES METADATA
195
DELIVERY
AI
HTTP OTHER
196
IMMERSION

FIGURE 1A

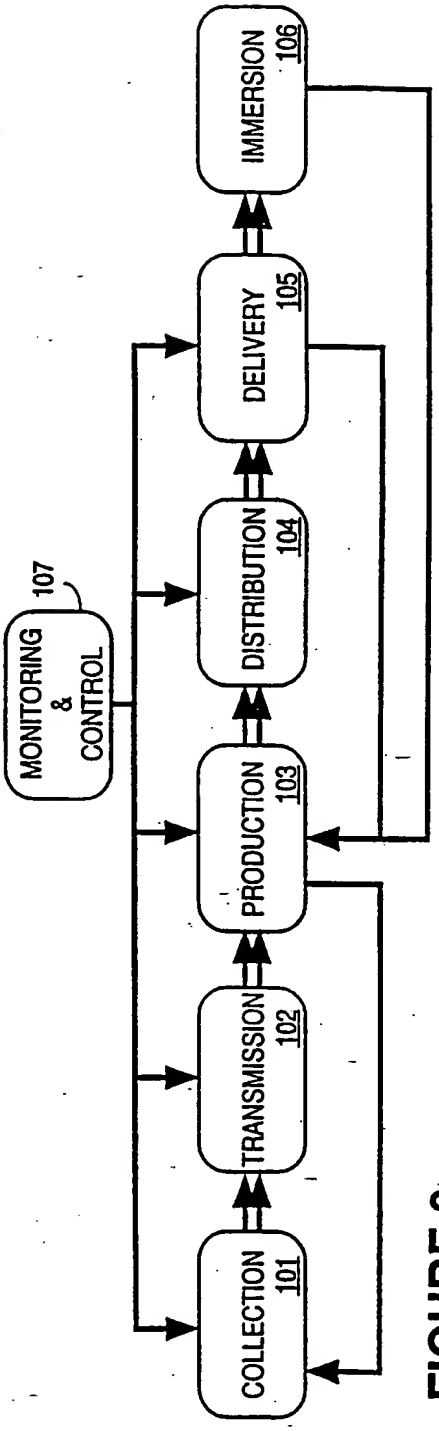
COLLECTION 101
TRANSMISSION 102
PRODUCTION 103
DISTRIBUTION 104
DELIVERY 105
IMMERSION 106
MONITORING & CONTROL 107

FIGURE 2

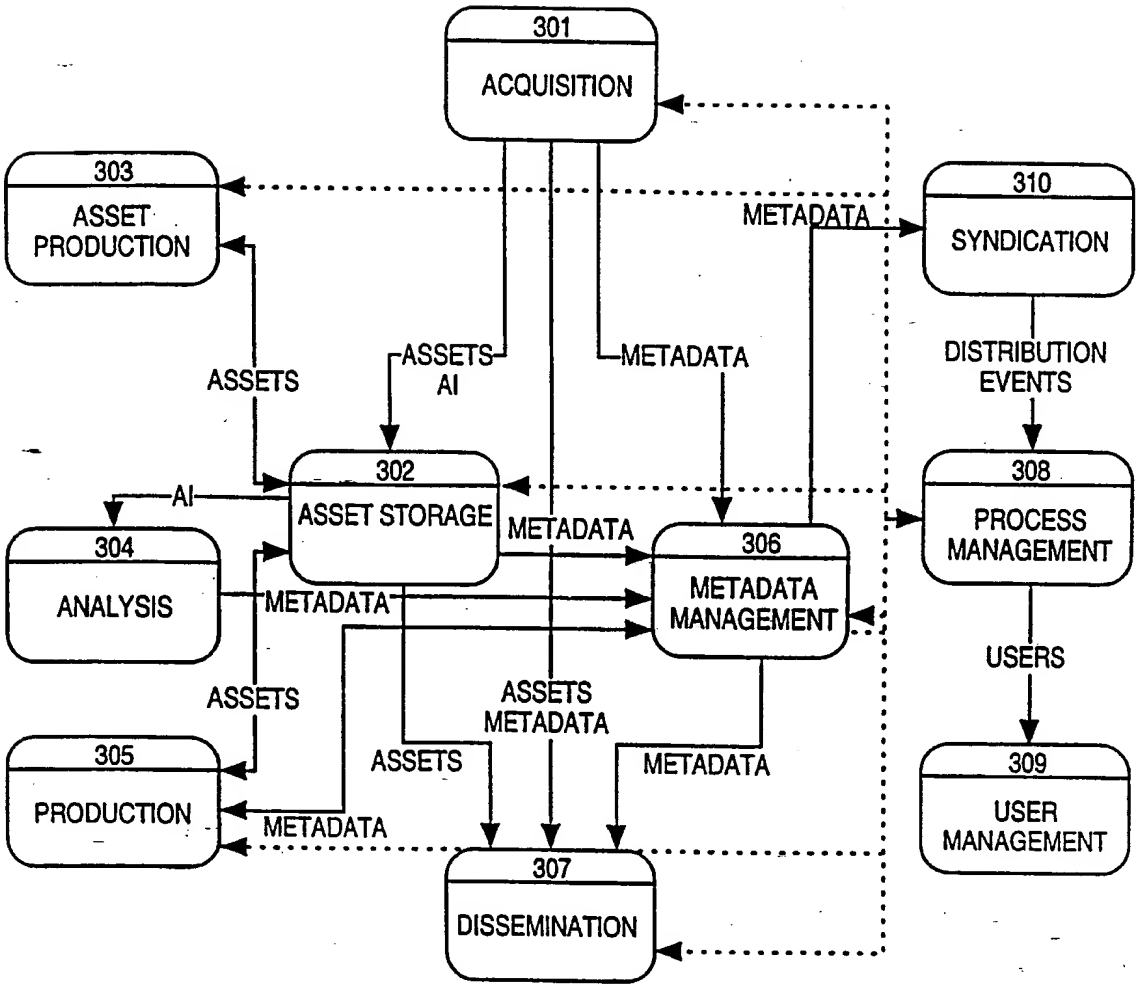
301
ACQUISITION
303
ASSET PRODUCTION
310
SYNDICATION
METADATA
ASSETS
ASSETS AI
METADATA
DISTRIBUTION EVENTS
302
ASSET STORAGE
AI
304
ANALYSIS
METADATA
306
METADATA MANAGEMENT
308
PROCESS MANAGEMENT
METADATA
USERS
ASSETS
ASSETS METADATA
305
PRODUCTION
ASSETS
METADATA
METADATA
309
USER MANAGEMENT
307
DISSEMINATION

FIGURE 3

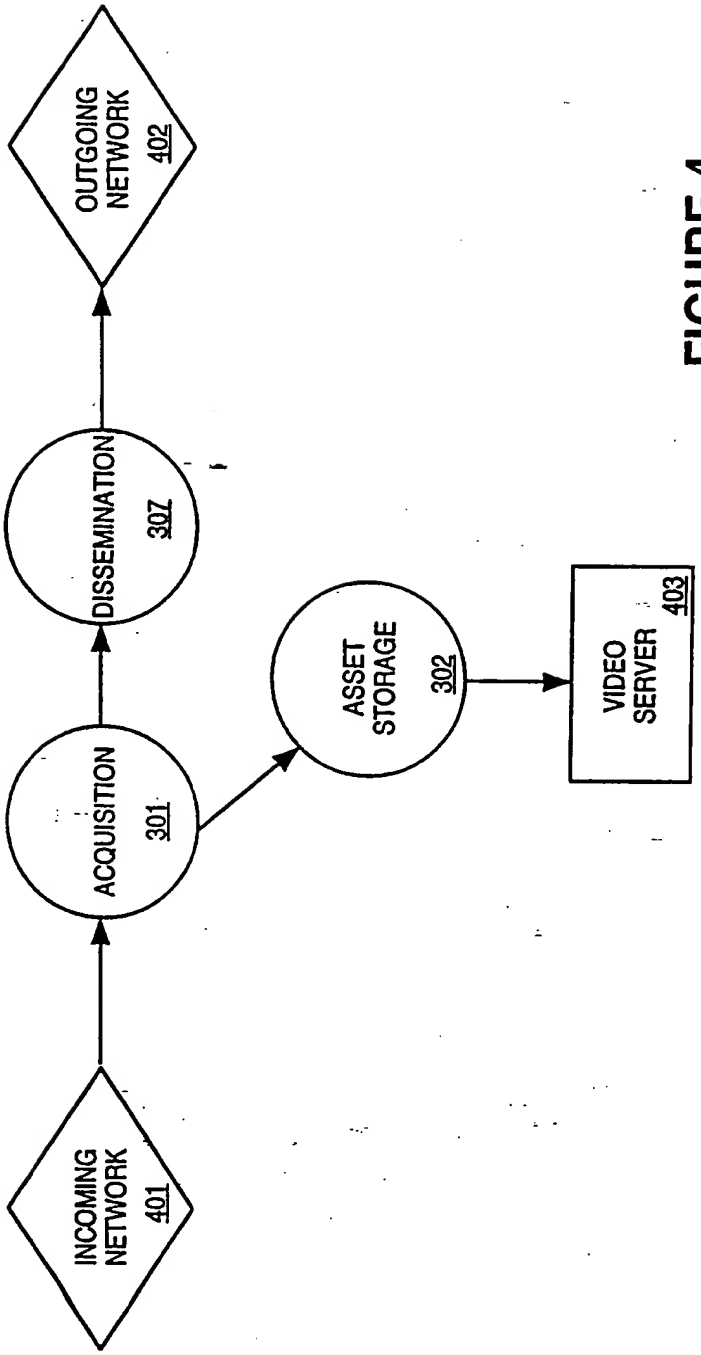
INCOMING NETWORK 401
ACQUISITION 301
DISSEMINATION 307
OUTGOING NETWORK 402
ASSET STORAGE 302
VIDEO SERVER 403

FIGURE 4

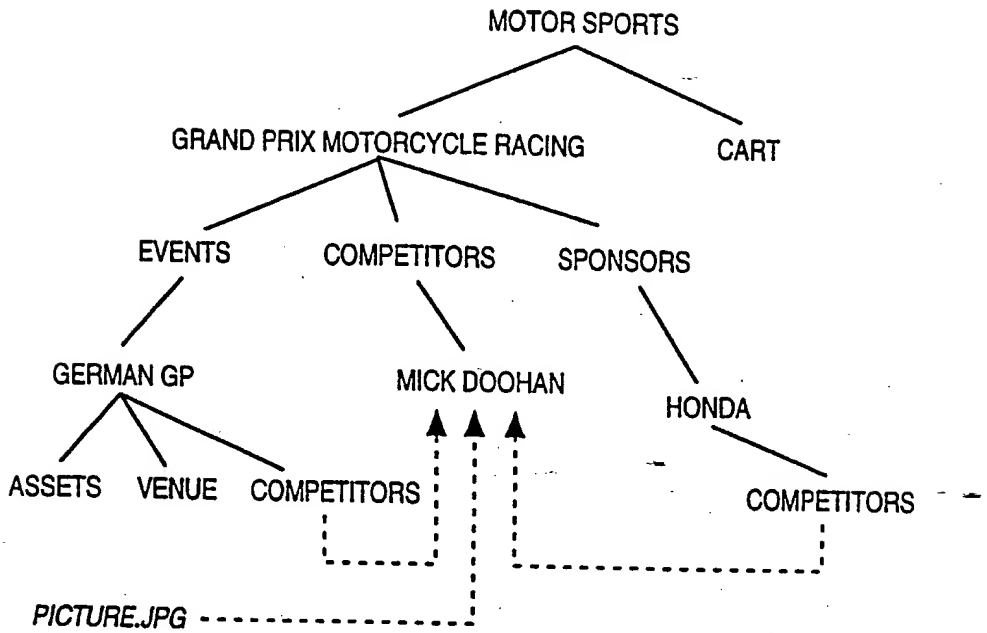

FIGURE 5

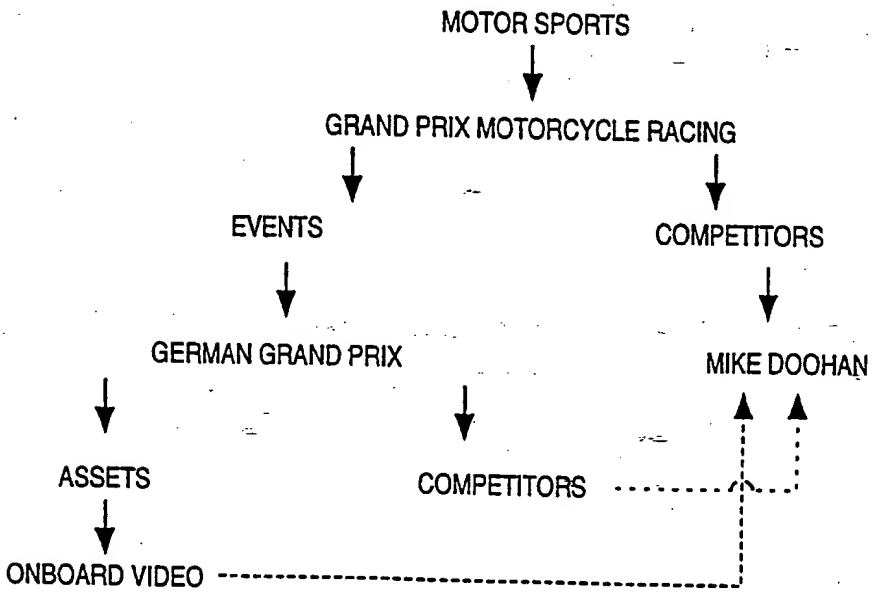

FIGURE 6

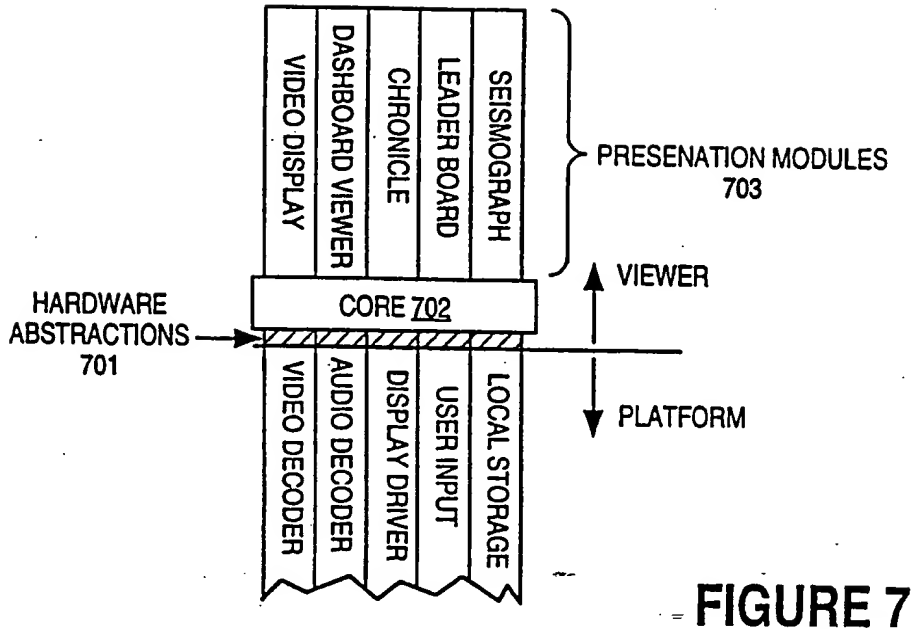

FIGURE 7

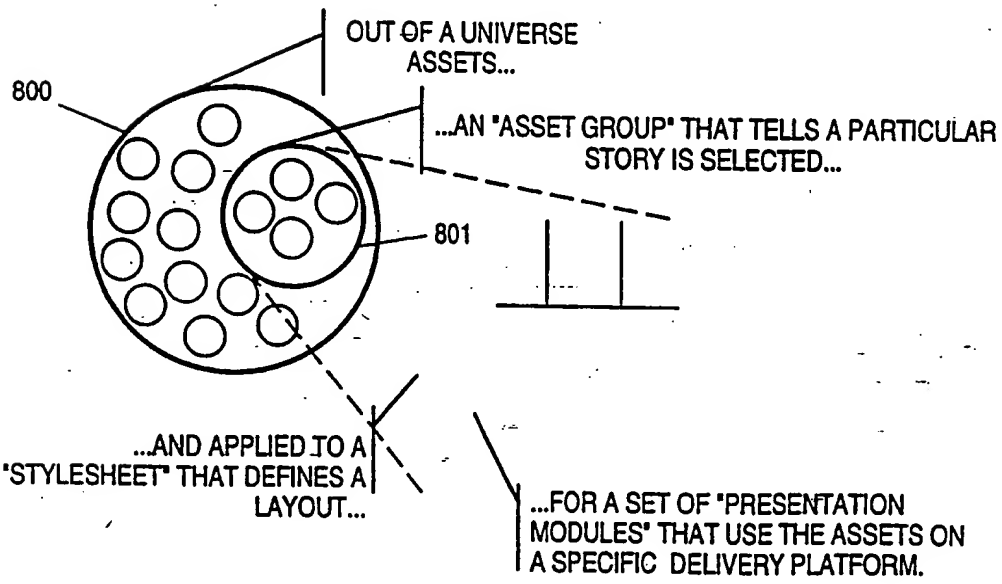

FIGURE 8

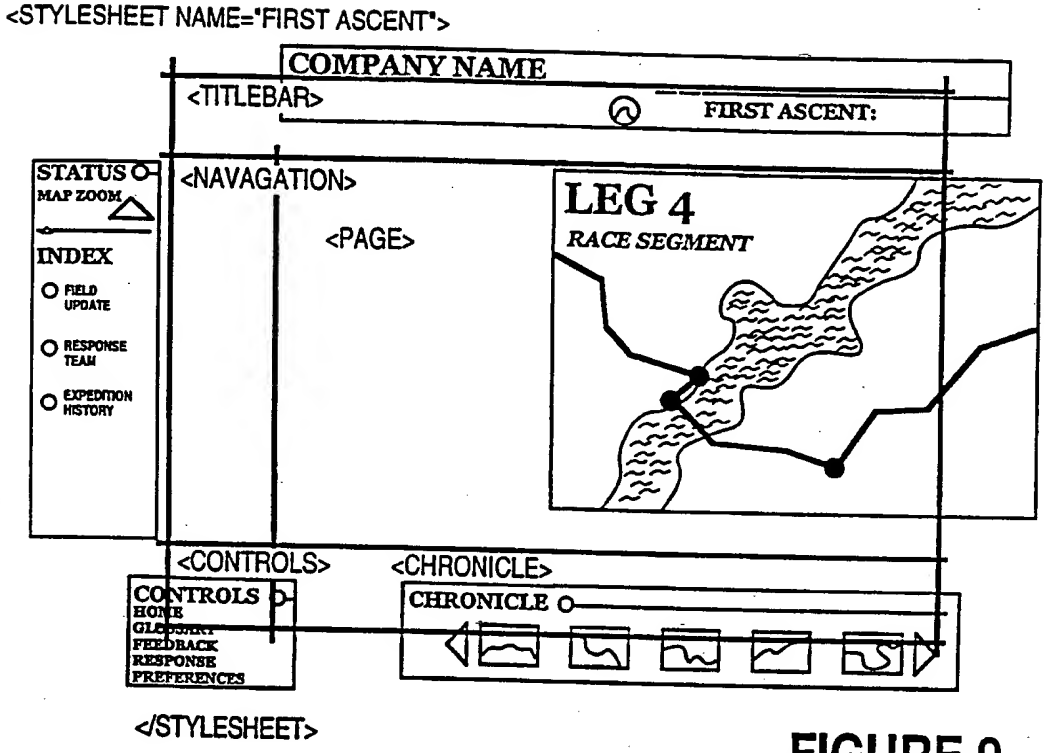

FIGURE 9

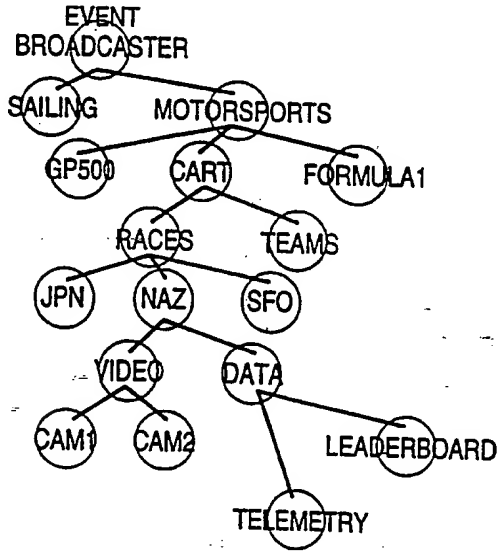

FIGURE 10

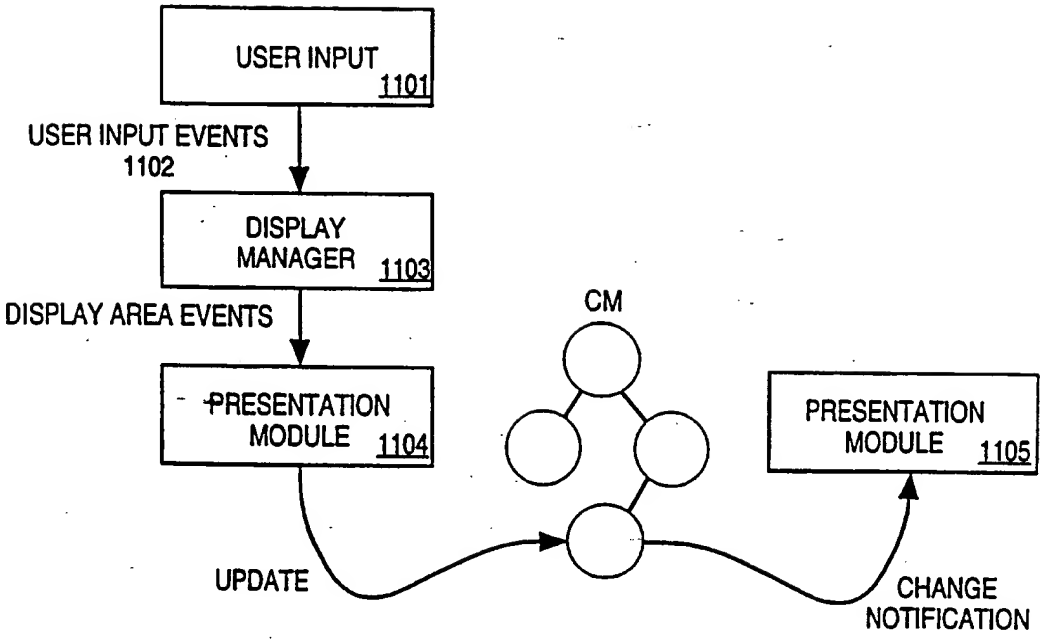
USER INPUT
1101
USER INPUT EVENTS
1102
DISPLAY
MANAGER
1103
DISPLAY AREA EVENTS
PRESENTATION
MODULE
1104
CM
PRESENTATION
MODULE
1105
UPDATE
CHANGE
NOTIFICATION

FIGURE 11

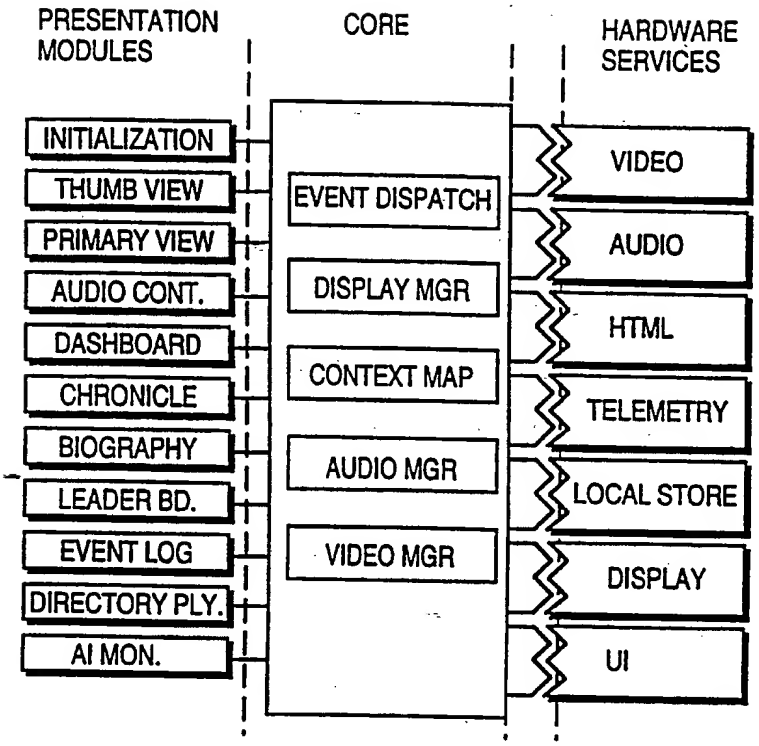
PRESENTATION MODULES
CORE
HARDWARE SERVICES
INITIALIZATION
THUMB VIEW
PRIMARY VIEW
AUDIO CONT.
DASHBOARD
CHRONICLE
BIOGRAPHY
LEADER BD.
EVENT LOG
DIRECTORY PLY.
AI MON.
EVENT DISPATCH
DISPLAY MGR
CONTEXT MAP
AUDIO MGR
VIDEO MGR
VIDEO
AUDIO
HTML
TELEMETRY
LOCAL STORE
DISPLAY
UI

FIGURE 12

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.